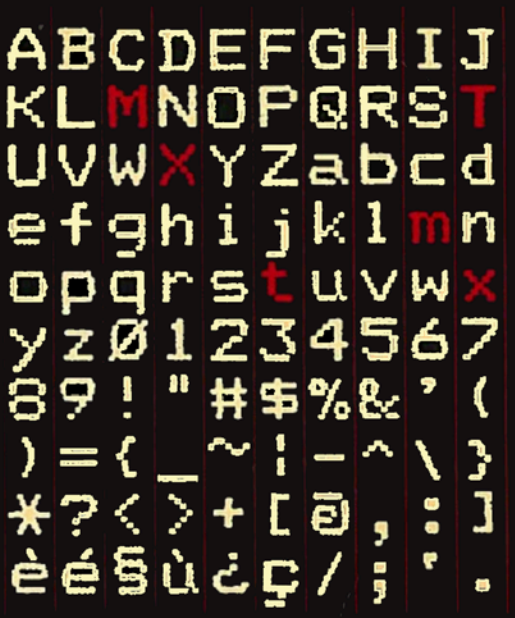


MEMOTECH

BASIC TUTOR, REFERENCE &
OPERATOR'S MANUAL

MTX
SERIES

MTX SERIES

OPERATOR'S MANUAL

Copyright Memotech Limited 1983

Brian Pritchard (Principal Psychologist, TRC, OXFORD).

CONTENTS

		Page
PART 0	Introduction	
Starting out		3
Beginning Basic		5
PART 1	MTX Series BASIC tutor	9
1	Programs and the tape recorder	9
2	Arithmetic expressions	19
3	Calculation order	21
4	Strings	25
5	The Printer	27
6	Storing information: Variables	29
7	Program writing	33
8	Using data	39
9	Entering data	43
10	Branching programs: Making decisions	45
11	Programs within programs	51
12	Structuring your programs	53
13	More branching programs	57
14	More about variables	59
15	Sorting	61
16	Multi-Dimensional arrays	65
17	Formatting with PRINT	69
18	Mathematical functions	71
19	String functions	73
20	Simple games and random numbers	75
21	Matrices	79

PART 2 NODDY 83

PART 3 GRAPHICS 97

PART 4 SOUND 123

PART 5 MTX ASSEMBLER 129

REFERENCE SECTION 135

SOFTWARE APPENDICES 173

1 ASCII Code Table 174
2 Control and Escape sequences 175
3 Error Messages 176
4 Numeric Keypad 178
5 System Variables 179
6 Function Keys 183
7 Colour Table 184
8 Sound Tables 185
9 Absolute Directions 188
10 Flowchart Conventions 189

GLOSSARY 190

MTX TECHNICAL APPENDICES 201
1 Overall Description 202
2 Technical Specification 203
3 System Bus 208
4 System Block Diagram 209
5 Electronic Schematics 210
6 Video Display Processor 216
7 Sound Generator 239
8 Hardware Memory Maps 245
9 Input-Output Ports 248
10 Parallel Printer Interface 251
11 Parallel I-O Port 252
12 Memotech DMX80 Printer Connector 253

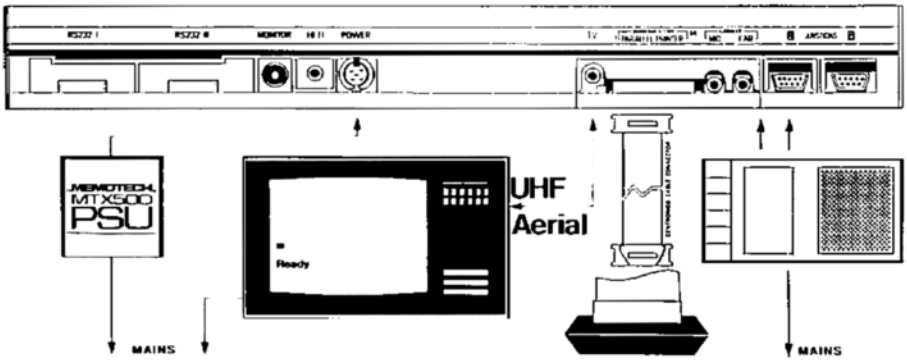
PART 0

STARTING OUT

As you will already have discovered, the MTX package contains much more than a computer. In the box you will find:

1. Your MTX Series computer.
2. An MTX power supply unit.
3. Cassette recorder leads.
4. UHF/VHF Television lead.
5. MEMOTECH Demonstration, Head cleaner and Blank tapes.
6. Snap-in Cartridge port cover. (This may be attached).
7. CONTINENTAL SOFTWARE have provided two complimentary games tapes and supporting literature.
8. Guarantee and User Registration card - Don't forget to fill this in and send it to us as soon as possible.

All you need to provide is your television, (and a cassette recorder to LOAD the tapes provided).



The first step is to make sure your system is properly connected; see the diagram above. After connecting the computer to the aerial socket on your television, tune the television until it gives a clear picture. Most televisions have an easily accessible set of buttons, one for each channel with a number of additional unused channel buttons. Select one of these and with your computer switched on adjust your picture until a clear blue background is found with 'Ready' in the bottom left hand corner.

Your MTX computer has been designed as an all purpose, hard wearing computer. As such, in normal working conditions you can expect perfect performances. It is also a precision electronic instrument and needs your care. Avoid working in conditions where liquids can be spilled into the computer or where the computer can be affected by excess heat.

To keep your MTX clean, we recommend that you use a clean chamois leather for best results.

The MTX computer manual has been written to enable a novice to get started on the computer using MTX BASIC. At the same time we have provided the expert programmer with detailed technical information on the use of MTX BASIC, GRAPHICS, NODDY and the ASSEMBLER.
If you are a beginner, start at the introduction and work your way through the manual. We have included a glossary of terms and a detailed reference section to help you understand computer jargon and how the command words and functions work. For the more experienced user a quick scan through the reference section will indicate the differences between MTX BASIC and other versions.

BEGINNING BASIC

The course you are about to begin has been written with the complete novice in mind. Even if you have learned BASIC before it will be useful to familiarise yourself with the rules which decide how MTX BASIC must be constructed. Throughout this course you will find a series of exercise programs which gradually increase in difficulty as you increase in confidence. By the time you reach the last section you will be a very skilful BASIC programmer.

COMPUTERS

A DIGITAL COMPUTER stores large amounts of information called DATA and is capable of carrying out simple tasks on that data at very high speeds. To produce the required results, the computer user gives the computer a set of instructions called a PROGRAM. Each of these instructions has to be set down in a precise way for the computer to understand what it has to do. This is done by using a computer language. Your MTX computer has incorporated into its structure four languages: MTX BASIC, Z80 ASSEMBLER, NODDY and MTX GRAPHICS.

NODDY is a new language which you probably have not heard about yet. The best way to describe what NODDY does is to compare it with LOGO. Noddy is to text as LOGO is to graphics.

MTX GRAPHICS is a comprehensive GRAPHICS package which allows you to set up LOGO and also design complex graphics programs from BASIC.

You will find NODDY and MTX GRAPHICS are dealt with in PARTS 2 and 3. NODDY does not require you to have any knowledge of what follows next, so if you wish, you can turn there at the end of this section.

The programming language you will probably want to tackle first is BASIC, or the Beginners All-purpose Symbolic Instruction Code. This begins at CHAPTER 1. It is suggested that you understand each section before starting the next. In this way it is unlikely that you will get hopelessly lost, and if you attempt all the exercises and try to think of some for yourself, you will quickly master the language.

Each section in the course is organised in the same way. The OBJECTIVE describes the problem you are about to attempt and explains what you will learn from its successful completion. In the sections where there are programs you may find partially completed FLOWCHARTS for you to finish. You will also find a sample program to try, and an exercise to make sure you understand the section.

The MTX keyboard is divided into three parts. The large block on the left is a standard alpha numeric keyboard. If you are not familiar with keyboards, type in some simple sentences to see what the keys do. If you by chance use a command that the computer understands, it may display an error message on the bottom line of the screen, but don't worry! It is impossible for you to do any harm to your MTX computer in this way. Discover what the SHIFT and ALPHA LOCK keys do, and get the feel of typing words on the keyboard. Each of the keys when held down will repeat its function until you lift your finger. This is called AUTO REPEAT.
The RET key on the right is used to tell the computer to accept whatever you have typed.
You can type on four lines at the foot of the screen when working in BASIC. To allow you to fill more of the screen we are going to teach you how to use a NODDY page.

Type NODDY and press the <RET> key

Noddy> will appear at the bottom of the screen and the computer will wait for you to give the page a title. In this case we shall use AA.

Type AA and press the <RET> key.

AA will appear at the top of the screen.

Do not press the <RET> key until you have finished the page. Just carry on typing in sentences and familiarising yourself with the keys. When you run out of space on page AA press the <RET> key and start a new page BB. If you press keys which do things you cannot understand, don't worry, reset the machine by using the RESET keys described in the next paragraph and start again.

The alpha numeric block has some additional keys which are not found on a typewriter. For example, you will see that there are two unmarked keys on either side of the space bar called the RESET keys. If you press both at the same time, the computer is cleared and will behave as if it has just been switched on. This can be very useful if you have made an error and wish to start again. It is not so convenient, however if you are part of the way through a large piece of work and carelessly lean on the keyboard. The other two unusual keys are the ESC (ESCAPE) and CTRL (CONTROL) keys. These are sometimes used during the running of programs and will be dealt with later in the manual.

The second keypad is called the EDITOR and NUMERIC keypad which consists of twelve keys. It has been designed to help you edit information on the screen, to correct your mistakes, to add in new information or to use in games as a keypad joystick. Type some information on to a NODDY screen as you did earlier and then use the keys as they are described below to change what you have written. Use the keys with arrows to move the cursor around the text. Use the INS key to insert text and the DEL and EOL keys to erase text. Do not use BRK yet.

Keys to move the cursor:

Moves the cursor one space to the left over the text.

Moves the cursor one line up over the text. (As in EOL the cursor up key does not operate in the BASIC EDIT screen because the computer only recognises one line, and so you can only move left or right and not up or down.

The cursor down key moves the cursor one line down over the text. (See the note in EOL and above.)

Moves the cursor one position to the right over the text.

TAB — Moves the cursor across the screen from left to right over your text in leaps of eight characters. It is a useful key when moving forward quickly over a lot of text.

HOME — This key takes the cursor back to the beginning of the screen you are working in.

Keys to erase:

DEL — Deletes the character over which the cursor is positioned.

CLS — This key clears the screen you are editing. It is a useful key if you wish to start the page again. However, as with the reset keys it can easily be pressed through carelessness resulting in unintentional loss of work.

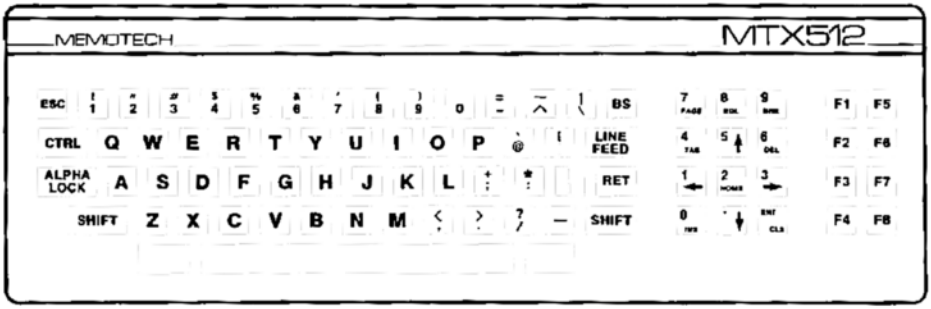
INS — Allows you to add text into a line without affecting information you have already typed. Simply press the INS key and type in the word or character you need. You will see that the text to the right moves along to make space for the new characters. INSERT will stay switched on until it is switched off by pressing the insert key again.

EOL — Deletes all the characters from the cursor position to the end of the line you are working in. (NB when you are working in BASIC the information which appears as four lines on the EDIT screen is in fact treated as a single line and therefore EOL deletes all information after the cursor.)

Keys to control programs:

BRK — Is used to stop a BASIC program operating and return you to Ready. This is dealt with later and should not concern you now.

PAGE — Is used to interrupt listings of programs and to switch between page and scroll modes. These terms will be dealt with in the relevant sections of the manual and you need not worry about them now.



The third keypad is called the FUNCTION keypad. These keys are in addition to the normal keys and are available for the user to tailor them to his own requirements. For example, they may be used to control LOGO or if you design a game they can be used to operate it. Though you can only see 8 keys each key will give a different function when used with the shift key allowing 16 user functions. The alpha lock key does not operate on this keypad. The method of defining the function of the keys is dealt with in the Software Appendix.

Some of the words in the course are going to be new to you. Most of them will be explained as you go along, but occasionally you may read a word you don't understand. If this does happen, have a look at the glossary for an explanation.

You are now ready to begin CHAPTER 1 on Page 9.

PART 1

BASIC TUTOR

CHAPTER 1.

PROGRAMS AND THE TAPE RECORDER

OBJECTIVE: The objective of this first section is to LOAD, RUN, SAVE and VERIFY programs with your computer.

RUNNING A PROGRAM

Make sure that you have connected the computer properly as described in PART 0. To put a program into the computer's memory from a cassette you use the command LOAD. Connect the cassette recorder to your MTX computer as shown in the diagram. Set the cassette volume to about 3/4 and then type LOAD on your keyboard. The word will appear on the screen. Now press the SPACEBAR and type the name of the game you are about to play in inverted commas. If we are loading a game called CHESS the screen would appear like this:

```
LOAD "CHESS"
```

If you do not know the program name you can type LOAD "".
If you do this, the computer will accept the next program on the tape. Though this method works, it is good practice to use the full version and successful loading is more likely if you have specified the name.

Press the RETurn key on your keyboard followed by the play button on your cassette. The computer will work out how many characters are to be loaded and will count them as they are placed in the working memory. When loading is complete, the screen will appear as follows if loading has been successful.

```
LOAD "CHESS"
FOUND CHESS
LOADING
■

Ready
```

To RUN the PROGRAM you have just loaded, you must tell the computer to carry out the instructions you have placed in its working memory. To do this you give it the command **RUN**. Type RUN on the keyboard and press the RETurn key.
Some programs run as soon as they are loaded. In this case it is not necessary to type RUN.

You may wish to RUN the PROGRAM again, in which case you do not have to reload it since it is already in working memory. It is useful, however, to be able to clear the screen of any information from the last RUN. To do this you use the command CLS (Clear Screen). Type CLS and press the RETurn key. The screen will now be clear, and the CURSOR will be positioned in the HOME position. Then as before, type RUN and press the RETurn key and the PROGRAM will RUN again.

The CLS key is different from the command CLS. The CLS key can be very useful and should be remembered because it can be thought of as a sort of 'panic button'. If you think at any stage that the computer is getting the upper hand, press the CLS key followed by RETurn and the edit screen will be cleared putting you back in charge.

The edit screen is the part of the screen which displays what you type on the keyboard.

This is explained further a bit later.

COPYING a PROGRAM

The LISTing that follows is an example of a small program for you to copy. Don't worry if it doesn't make much sense at the moment, the object of the exercise is to show you what an MTX BASIC program looks like, and accustom you to the computer.

If the computer has already been used to play a game or run a program, you will have to remove the contents of the computer's memory before you try to copy the LISTing below. To do this you use the command NEW which tells the computer to forget what it currently has in its memory in order to accept a new program. Type NEW and press the RETurn key.

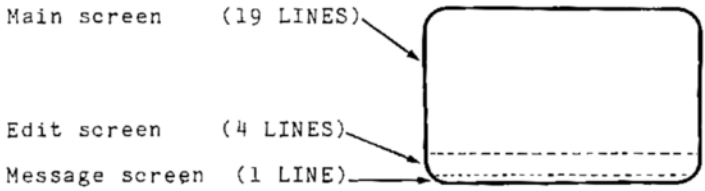
```
10 REM COPY PROGRAM
20 PRINT "WHAT IS YOUR NAME"
30 INPUT N$
35 PRINT:PRINT
40 PRINT "WHAT IS YOUR AGE"
50 INPUT A
60 CLS
70 PRINT N$;" IS";A;" YEARS OLD"
80 PRINT:PRINT
90 PRINT "AGAIN"
100 INPUT M$
110 IF M$="Y" THEN GOTO 10 ELSE STOP
```

Type the LISTing above, remembering that it must be copied exactly as it appears and press the RETurn key at the end of each line. Check that each line is identical to that in the LISTing. If it is not, it is likely that the program will not work. If it doesn't work and you can't see why, type NEW <RET> and start again.

Since we shall use the RETurn key so often, we shall usually refer to it as <RET>.

The BASIC screens

You will notice as you type the program LISTing into the computer that the screen is divided into three sections. There are 24 lines on the screen, split up as follows:



Information when first typed is placed on the EDIT screen. When you are satisfied that the line is correct you ask the computer to accept the line by pressing <RET>. The line is then moved to the MAIN screen as part of the program. If you have made a mistake the line may not be accepted by the computer and an ERROR MESSAGE may appear in the Message screen. The CURSOR will move to the position on the line where the error was found. This type of error occurs when you have typed in something which the computer does not understand.

An example of an ERROR MESSAGE is given below:

```
Mismatch
```

This tells you that you have made an error and that it is a SYNTAX error. This just means that what you have typed in is not acceptable as part of the language BASIC, and the computer does not know how to handle the line. The error message above would occur if you typed, for example:

```
110 IF M$="Y" THEN GOTO "10" ELSE STOP
```

Try it and see for yourself. The error message appears after you press <RET>. The reason is simply that "10" is not recognised as a number.

There are a number of ways to correct mistakes like this; the simplest is to type directly over the parts that are wrong. In the above example the DELete key could be used to take out the inverted commas. This is achieved by moving the CURSOR to the first character to be deleted, using the arrow keys in the editor keypad, and then pressing the DELete key. When you think the line is correct, press <RET>. The edit keys on the centre keyboard help you correct errors like this.

LISTING YOUR PROGRAMS

To see what you have written, LIST the program by typing the command LIST <RET> and you will see your program listed in its correct order. It is also possible to LIST sections of your program, by using a variation of the command LIST. This is more useful for editing longer programs, but can be demonstrated here.

Type LIST 20,40 <RET>

and you can see that lines 20, 30,35 and 40 appear on the main screen. Similarly,

LIST 30

displays from line 30 to the end of the program.

As this is a short program, the whole listing is in view. However, many programs including some that you will design later in this manual have more lines than are available on the screen. To look through such programs the PAGE key is used to interrupt the listing. As the program scrolls up the screen the first press of the PAGE key stops the scrolling and the second restarts it. It is very useful to scan programs in this way to look for obvious errors.

Now you have input the program correctly, you are ready to try it out. There is no need to LOAD the program since it is already in the computer's MEMORY.

Type RUN <RET>

RUN TIME ERRORS

If you have made no mistakes, the program will RUN successfully. However, when you think you have completed the program there may still be errors. These do not appear as SYNTAX ERRORS and are called RUN TIME ERRORS. They occur, as their name suggests, while the program is trying to RUN, and are caused when you have entered the correct commands, but you may not, for example, have given the correct information for the computer to carry out the command. If for example the name and age program was altered so line 110 read:

110 IF M$="Y" THEN GOTO 120 ELSE STOP

The command is correct and the number 120 is of the type that the computer expects, but there is no line 120 in the program, and so the computer cannot continue. The line should read:

110 IF M$="Y" THEN GOTO 10 ELSE STOP

If you try to run the program with this incorrect line an error situation will occur and the computer will not know what to do with the incorrect instruction, so it will not carry on with the RUN. The computer will tell you the reason for being unable to continue and the line with the error will be displayed. In the case of the incorrect line 110, the message

```
No line
```

would appear. Since we have a line 110, we know that the error message must refer to the 120 in the GOTO statement. RUN TIME ERRORS are edited in much the same way as the SYNTAX ERRORS described earlier.

There are a number of alternative editing methods it may be useful for you to understand. You may for example wish to type a whole line again; the computer will accept your most recent attempt at a line, wherever it occurs in a program, so if you type

```
50 INPUT A
```

even at the end of the program, the computer will look for line 50 and replace the earlier version.

If you wish to delete a whole line, you need type only the line number; the computer will understand that you do not want this line in the program, and will delete it accordingly.

A complete list of ERROR MESSAGES appears in the Software Appendix. Although editing is a useful way of correcting your mistakes, in practice it is not a good idea to continue to EDIT a program where it is obvious there are a lot of errors. In this case it is better to start again, as you will often find it is difficult, if not impossible to get heavily edited programs running at all.

When you are happy the program is correct and running perfectly, you may wish to SAVE it.

SAVING A PROGRAM

To SAVE a program it is necessary to set up your system in the same way as you did for LOADING a program.Firstly insert a blank cassette into your recorder. We recommend that you use C15 or C30 cassettes and record only ONE program on each side, so that your work is always kept organised. You will be surprised how easy it is to forget which programs are on which tapes, and whereabouts on the tape they occur, etc. This can be overcome to a certain extent if your cassette recorder has a tape counter but always remember to keep a strict log of which programs you have SAVED and where they are on your cassettes. Set the recorder to record and use the pause button if you have one, to hold the tape until the computer is ready.

To SAVE a program you use the command SAVE.

Type SAVE "filename" <RET>

Press <RET> after the tape has been started. In the example we could use the filename "AGE" and as it is your first version, the SAVE instruction could be:

```
SAVE "AGE 1"
```

When you have used a filename in this way, the program can be recognised and loaded using this name. It is advisable, therefore, to use meaningful names for your programs, and write them clearly on to the tape label.

When the computer has finished saving the program, the screen will look like this:

```
SAVE "AGE 1"

Ready
```

When saving has finished, it is possible for an error to have occurred and so MTX BASIC has the command VERIFY to allow you to check that the program has been properly recorded. Rewind the tape to the beginning of the recorded program and type:

```
VERIFY "AGE 1" <RET>
```

Now play the tape, and the computer will check each character recorded against those in its MEMORY. When verification has finished, this is how the screen will appear:

```
VERIFY "AGE 1"
FOUND AGE 1
VERIYFYING

Ready
```

You have now successfully SAVED and VERIFIED your first program!

If an error has occurred, 'Mismatch' will appear in the message screen. The program is still in MEMORY and so you can try saving it again.

COLOUR

Whilst in the editing or command mode, the screen will always be blue and white, but when a program is running, the colours may be changed.

The background colour is called the PAPER colour and the foreground is called INK. Just think of the screen as a writing pad and all becomes clear.

Type in the following program remembering to press <RET> at the end of each line.

```
 10 PRINT:PRINT
 20 PRINT:PRINT
 30 PRINT "&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&
&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&
&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&"
 40 PAPER 7
 50 INK 1
 60 PAUSE 8000
```

Type RUN and for about eight seconds the colours will change.

Each of the 16 colours the computer can produce has a unique number as in the colour table below. To see each of the colours edit the program like this:

Type EDIT 40 <RET>

Line 40 will appear

40 PAPER 7

Move the cursor to a position over the 7 and type 6 followed by <RET>. When you RUN the program now the background will change to a dark red.

Edit the program using the table below changing PAPER and INK.

COLOUR TABLE

0 TRANSPARENT
1 BLACK
2 MEDIUM GREEN
3 LIGHT GREEN
4 DARK BLUE
5 LIGHT BLUE
6 DARK RED
7 CYAN
8 MEDIUM RED
9 LIGHT RED
10 DARK YELLOW
11 LIGHT YELLOW
12 DARK GREEN
13 MAGENTA
14 GREY
15 WHITE

NB Make sure that your television or monitor are perfectly tunec to run these programs.

CHAPTER 2.

ARITHMETIC EXPRESSIONS

OBJECTIVE: To use the computer as a simple calculator to add, subtract, multiply and divide.

We are now going to have a look at the PRINT command and use it to place on the screen the results of simple calculations. The PRINT command writes on the screen any information that immediately follows it.

Type PRINT 21 <RET>

This will write 21 on the next line.

Now try some other numbers.

If you wish to place numbers across the page in columns then the numbers are typed with a comma after each one.

Try this:

PRINT 3,4,5

Each number is placed at the next available TAB position. The TAB positions are spaced eight characters apart across the screen.

Now try:

PRINT 2 + 2 <RET>

The answer 4 will appear on the screen. The PRINT command will send the result of the calculation to the screen. Similarly the computer will work out subtractions using the minus sign (-) as in:

PRINT 7 - 4 <RET>

(NB The minus sign is on the same key as "=" on the top row and is not the underline next to the right hand shift key.)

The signs for multiply. and divide, however, are slightly different, but these will become second nature in no time. For example:

PRINT 8 * 3 will multiply 8 by 3 and the answer 24 will appear on the screen.

PRINT 6/2 will divide 6 by 2 and the answer 3 will appear on the screen.

The addition 2 + 2 is an example of an ARITHMETIC EXPRESSION. In each of the examples above the PRINT command is being used to print the value of the arithmetic expression which follows. In EXERCISE 1 you are asked to EVALUATE (find the value of) the arithmetic expressions.

//

EXERCISE 1 ARITHMETIC EXPRESSIONS

Use the PRINT command to evaluate the following arithmetic expressions.

1) 2 + 2	2) 15 + 35	3) 288 + 397	4) 7945 + 3538
5) 7 - 4	6) 28 - 14	7) 654 - 289	8) 7986 - 3572
9) 8 x 3	10) 16 x 4	11) 244 x 6	12) 387 x 28
13) 6 ÷ 2	14) 60 ÷ 5	15) 288 ÷.06	16) 1080 ÷ 72

//

Try some calculations of your own. Notice that if the solution is not a whole number then your MTX automatically works out the answer as a decimal.

CHAPTER 3.

CALCULATION ORDER

OBJECTIVE: To introduce the order in which arithmetical expressions are evaluated.

In CHAPTER 2 you used the computer as a simple calculator. In this section you will learn how to work out more complicated problems.

When a mathematical expression is more complex or uses squares or square roots, we have to write it in a form that the computer can easily understand. Look at this example:

3^2 is written

3^2 in MTX BASIC

The power 2 is written ^2. The answer to the above example is of course 9.

//

EXERCISE 2 POWERS
Using PRINT, solve the following examples:

$9^2 = ?$ $20^2 = ?$ $10^3 = ?$ $2^{12} = ?$

//

The calculations in CHAPTER 2 and those above require only one operation. Now try this calculation:

```
2/3*6
```

The answer is 4. To evaluate this expression the computer worked from left to right and followed a specific order. Operations are always performed in the following order:

Sign	Operation	Example
^	Exponentiation(power)	2^2 = 4
* and /	Multiplication and Division	3*2 = 6 6/3 = 2
+ and -	Addition Subtraction	3+2 = 5 3-2 = 1

Each time the computer is asked to evaluate an expression it works from left to right and uses this order of calculation. Look at this example:

$$3 + 7 \times 5^{2} + 4 \div 2 - 6 \times 2$$

To PRINT the answer type the following:

```
PRINT 3 + 7 * 5^2 + 4/2 - 6 * 2
```

This gives an answer of 168. The computer evaluates the expression using the following steps:

```
3  +  7 * 5 ^ 2 + 4 / 2 - 6 * 2     STEP 1 Exponentiation

             25

3  +  7 *   25  + 4 / 2 - 6 * 2     STEP 2 Multiplication
                                           and Division
       175          2       12

3  +   175      +   2   -   12      STEP 3 Addition
                                           and Subtraction
            = 168
```

//

EXERCISE 3 CALCULATION ORDER
Break the following calculations into STEPS as above and then PRINT them on the screen to check your answers. The first one has been partly completed for you.

1) $8 \times 2^{2} + 8 \div 2^{2} - 9^{2} \div 3$

```
8 ? 2^2 + 8/2^2 -9?  /3

    ?       ?     ?

8 *  ?  + 8/? - ?/3

  32    +  ?  -  27

      = 7
```

2) $38 + 64 + 6 \div 3 - 6^{2} \div 3$

3) $16 + 18 \div 3^{2} - 2 \times 3^{2}$

4) $64 - 7^{2} + 6^{2} \times 2 - 2^{6}$

//

If the order of a calculation is altered then a very different answer is obtained. Look at the example we used earlier using PRINT:

```
2/3*6   Has the value 4

2/(3*6) Now has the value 0.111111111
```

This is because we have altered the order by using brackets.

The computer calculates the contents of brackets first.

Look at these simple examples.

If John had 5 apples and Mark had 3 how would they share them evenly?

Try these solutions:

```
5+3/2
```

or

```
(5+3)/2
```

The correct solution is of course the second one since you have to add up the number of apples first and then divide the total number of apples between John and Mark.

Mark has £2.60 and wishes to give John half. He owes Kate 30p and has to pay her back first. This calculation would be carried out as follows:

```
(2.60 - .30)/2 =
```

Mark's father offers to give him 6 times the amount he has remaining. This could be worked out like this.

```
6*((2.60 - .30)/2) =
```

Finally his mother offers to square (!!) the amount his father has given to him.

```
(6*((2.60 - .30)/2))^2 =
```

Brackets used in this way are called NESTED brackets.When you use nested brackets the computer works from the middle bracket outwards and then left to right keeping to the calculation order described earlier.

Use brackets to solve the problems in EXERCISE 4

//

EXERCISE 4 BRACKETS IN CALCULATIONS
Use PRINT to solve these problems:

1) At the races Harry starts with £10. He places £2 on the first race and doubles his stake. On the second race he loses £4 and then places all his remaining money on two horses, the first of which trebles his money and the second of which doubles it again. How much does he have after the final race?

```
((((10-2) + 2*2)-4)*3)*2
```

2) A farmer has two identical circular fields (radius 200 metres) and two identical square fields (side length 75 metres). He then buys another farm of exactly the same dimensions. Use nested brackets to work out the total area of both farms in square metres. (The area of a circle is taken to be (PI*R^2), where PI = 22/7 approx.)

```
(((22/7*200^2)*2)+((75^2)*2))*2
```

//

The calculation of square roots presents another problem for the computer. One way of solving the problem is to use the fact that a square root can be expressed as a fraction of a power. For example, the square root of 4, written as $\sqrt{4}$ in everyday language, is exactly the same as writing 4^(1/2). Notice here how brackets are used to ensure the calculation is done in the correct order.

The cube root of 16 ($\sqrt[3]{16}$) can be written 16^(1/3), and so on.

//

EXERCISE 5 SQUARE ROOTS
Use the PRINT command to calculate the following:

$\sqrt{9}$ $\sqrt{25}$ $\sqrt{81}$ $\sqrt{144}$

$\sqrt[3]{27}$ $\sqrt[3]{216}$ $\sqrt[4]{50625}$ $\sqrt[9]{512}$

//

CHAPTER 4.

STRINGS

OBJECTIVE: To use the command PRINT to send text to the screen in the form of simple STRINGS

The PRINT command is used not only for printing numbers but also for writing any information on the screen. Textual material (books, addresses, names etc.) consists of letters, numbers and spaces which have to retain their order each time they are printed. When the computer is given a set of numbers it places them in an order which it finds convenient to evaluate. If this happened to textual material then the output from the computer would no longer be readable. Text is, therefore, input in such a way that the computer does not alter the order of the information.

There are a number of ways to do this but in this section we will concentrate on the use of STRINGS, represented by letters, spaces and numbers etc. placed in inverted commas.

STRINGS are stored by the computer exactly as they are written. Use the PRINT command to place this example on the screen:

```
PRINT "JOHN WESTON"
```

Now PRINT your name.

STRINGs don't have a fixed length and may contain any of the characters recognised by BASIC with the exception of inverted commas (") since these denote the beginning and end of the STRING.

For example:

```
PRINT "ABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890!£$%&'()_=-"
```

//

EXERCISE 6 SIMPLE STRINGS
Use the PRINT command to write the following on the screen

Your Name
Your Address
Your Date of Birth
Your Occupation (Job, School etc)

//

Just as there are operations that we can perform on numbers such as addition, multiplication and division, there are also ways of manipulating STRINGS. The simplest operation is to join STRINGS together to form a longer string. We do this by using the '+' sign because of the similarity with mathematical addition but don't confuse joining with adding.

Type:

```
PRINT "TELE"+"VISION"
```

Obviously it is not possible to subtract, multiply, divide or raise STRINGS to powers. As you will see later, STRINGS can be manipulated in various ways using other special string functions.

CHAPTER 5.

THE PRINTER

OBJECTIVE: To print out information on a printer.

BASIC provides a number of ways to control your printer. By far the most useful is the command LPRINT. This command operates in much the same way as PRINT, but instead of sending information to the screen LPRINT sends information to the printer. The command LPRINT refers to the term LINE PRINTER, which describes the type of printer used by larger computers.

```
LPRINT "JOHN WESTON"
```

If you have connected the printer properly

```
      JOHN WESTON
```

will be printed out. If you have forgotten to switch on the printer or connected it wrongly then the MTX computer will wait until the printer is made ready, and so you do not necessarily have to start again if you have made a mistake. The BRK (BREAK) key can be used to return to 'Ready' and stop the printer at any time.

You may wish to print out information as you work, in which case the LPRINT command works in the same way for the printer as PRINT works for the screen. Thus:

```
      LPRINT "The answer to 2+2 is?"
      LPRINT 2+2
```

will print:

```
      The answer to 2+2 is?
      4
```

To print out a program you have input you may use the command LLIST. This command will send to the printer the program currently in the memory. It cannot send to the printer a file directly from your cassette. The procedure in that case would be to LOAD the program and then LLIST.

It is possible to print out part of the program in memory, this is done using the LLIST command and works in the same way as LIST:

```
LLIST 100,200
```

will print lines 100 to 200 on the printer.

See Reference Section PRINT LPRINT LLIST

CHAPTER 6.

STORING INFORMATION: VARIABLES

OBJECTIVE: To introduce variables and their manipulation using the LET statement.

In CHAPTER 2 we used the PRINT statement to do simple calculations. If you look at the set of commands below you will see there is an alternative way to evaluate expressions:

```
LET A = 6
LET B = 2
LET X = A+B
PRINT X
```

A, B and X are called variable names. When a variable name is used in a program the computer automatically reserves space in memory for information, and gives the space that name. In these statements the values of 6 and 2 are stored in locations named A and B respectively. The sum of these is then stored in a third location named X, and the contents of location X is then printed.

////////////////////////////////\\

EXERCISE 7 LET STATEMENTS
PRINT the values of X in each case as in the example above.

1)
```
LET X = 6*4^2
```

2)
```
LET X = (4-2)^4*(6-3)^2
```

3)
```
LET A=6*8
LET B=16/2
LET C=B*6
LET X=A/C
```

4)
```
LET A=64^0.5
LET B=(A-4)^2
LET C=B-A
LET X=(A*C)/B
```

5)
```
LET X=4
LET Y=X^2
LET X=(Y-X)/6
```

\\////////////////////////////////

As with the PRINT command the LET command can apply to text as well as numbers. When text is allocated to a location then the location name is followed immediately by a $ sign to tell the computer to expect a STRING as below:

```
LET A$ = "JANE"
```

JANE is a STRING and is, therefore, in inverted commas. If you type this and then ask the computer to PRINT A$,JANE will be printed on the screen.

Now type in

```
LET B$ = " "
LET C$ = "SMITH"

PRINT A$+B$+C$
```

Remember this is not an addition. The '+' signs tell the computer to join the STRINGS end to end. B$ in this case simply puts a space between the two STRINGS which contain JANE SMITH's name.

Use the LET command in exercise 8 to place the information about yourself into string locations and then use the PRINT command tc send the information to the screen.

//

EXERCISE 8
Use the LET command to locate the information below in
N$,A$,B$ AND J$.
Your Name
Your Address
Your Date of Birth
Your Occupation (Job, School etc)

//

This method of storing information in locations allows for only 26 separate locations since there are 26 letters in the alphabet. A$,B$,C$.....Z$.

The same letter (for example A) can be used to allocate a variable name to a STRING and a number; the STRING in A$ will be given a quite separate location from the number in A. You can also extend the number of locations by using additional letters and numbers in the variable name. In the example below all the variable names refer to different memory locations.

eg A$, AA$, A1$, A, AA, A1, NAME$, AGE, ADDRESS$, etc.

CLEAR

It is sometimes useful to CLEAR the variables in memory so that calculations can start afresh. Perhaps now you may wish to CLEAR locations to alter EXERCISE 8. To CLEAR the locations type the command CLEAR and press <RET>.

All variables are automatically CLEARed every time a program is RUN or edited without using the command CLEAR.

//

EXERCISE 9 SETTING THE CLOCK

Your MTX computer contains a clock which can be set to REAL TIME. It can be also be used as a stop watch by setting it to zero. To set the clock for half past twelve for example, type:

```
CLOCK "123000"
```

This has set the hours to 12, the minutes to 30 and the seconds to zero.

The STRING consists of six digits, the first two representing the hours, the second two minutes and the last two seconds.

To display the time as you are working, type:

```
     PRINT TIME$
```

This displays the whole clock.

Try this program demonstrating the clock. Remember to input the time as a six digit number.

```
10 INPUT "WHAT IS THE TIME? ";A$
20 CLOCK A$
30 CLS
40 CSR 0,0
50 PRINT TIME$
60 GOTO 40
```

//

See Reference section LET, CLEAR, CLOCK, TIME$, CSR

CHAPTER 7.

PROGRAM WRITING

OBJECTIVE: To introduce the method of designing programs, using flow diagrams, numbering and the use of AUTO and REM.

You are now ready to begin writing programs in MTX BASIC. As you will have realised when copying the program in CHAPTER 1 there are strict rules to be adhered to if your programs are to work. Also, as you may have realised they can become fairly complicated, and therefore, it is essential that you plan your program before writing it, and that you keep your work organised as you write.

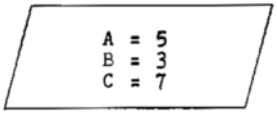
In this CHAPTER we are going to show you how each of these can be achieved by using FLOW CHARTS and REM statements.

FLOW CHARTS.

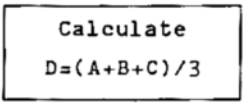
A FLOW CHART is a step by step description of the way in which a particular problem is going to be solved. Consider a problem for example, where JOHN had 5 apples, MARK had 3 and KATE had 7. How could we design a program to share these apples evenly? The information we need is placed into a table to show the VARIABLES required.

VARIABLES	
A	Marks Apples (5)
B	Johns Apples (3)
C	Kates Apples (7)
D	The number of apples each

The first step would be to tell the computer the number of apples each person had. When drawing flow charts, information to be input to or output from the computer, is placed in parallelograms. The first step then is:

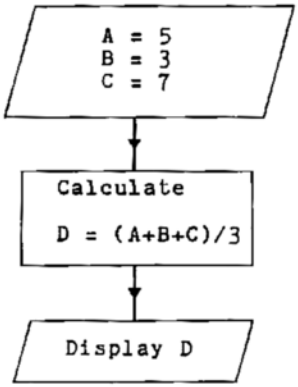


The second step is the calculation itself. We want to find the total number of apples (5+3+7) and divide this by the number of people (in this case there are three). Calculations of this type are placed in rectangular boxes. So this step looks like:



We have already discussed the order in which the computer does the calculations, in this example the division will be carried out first unless we place A+B+C in brackets.

The computer now has to be told to display the answer and since this is output, the instruction on the flow chart is placed in a parallelogram. The completed diagram now looks like this:



//

EXERCISE 10 FLOW CHARTS

1) Complete the flow chart below.
This flow chart shows how to design a program to convert test scores into percentages. The test has 25 questions and the number of correct answers are:

a)24 out of 25
b)19 out of 25
c)20 out of 25
d)15 out of 25

READ ?

CALCULATE
PERCENTAGE = ?

DISPLAY ?

Draw flow charts to solve these problems.

2)Petrol costs £1.89 per gallon. How much would you pay to buy:

a)5 gallons, b)7.25 gallons, c)11.68 gallons.

3)A gallon of petrol costs 4.54 times more than a litre of petrol. At £1.89 per gallon, how much would the following cost?

a)24 litres b)36 litres c)42 litres

//

WRITING A PROGRAM

A BASIC program is a series of instructions which are given to the computer in a language which it understands. Each instruction is generally placed on a new line, and must follow exactly the format which is required by BASIC. The instructions are then carried out in the order you have specified by numbering each line. Line numbers can range from 1 to 65536. Each line in BASIC must begin with a whole number. There can be any interval you like between the numbers and it is usual to write programs with an interval of at least 10 so that lines which have been forgotten can be included. Look at this example of the program written to share apples:

```
10 REM SHARING PROGRAM
20 LET A = 5
30 LET B = 3
40 LET D = (A+B+C)/3
50 PRINT D
35 LET C = 7
```

Line 35 was deliberately left out at first to demonstrate that it can be added at the end. This does not affect the working of the program since the lines are run in line number order and if you input this program you will see that it works perfectly well. Tidying up the program by putting all the lines in their correct order can be done by using the command LIST.

The REM statement in line 10 is the title of the program. The computer ignores anything that immediately follows the word REM They are used to REMind you of anything you think is relevant to help you remember how you structured your program, and as you can see in the rewritten program below, the REM statements refer to each box in the flow chart.

```
10 REM SHARING PROGRAM
20 REM LOCATING QUANTITIES OF APPLES
30 LET A = 5
40 LET B = 3
50 LET C = 7
60 REM CALCULATE SHARE
70 LET D = (A+B+C)/3
80 REM PRINT SHARE
90 PRINT D
```

Remember that flow charts when used in conjunction with REM statements keep you organised, so use them as much as possible.

REVISION

Try to write programs for the flow charts you completed in Exercise 10.

When you have completed a program do not forget that you have to tell the computer to make it work with the command RUN. You can only work on one program at a time, so remember to use NEW before starting your next exercise. Experiment with CLEAR, SAVE, LOAD, LIST and VERIFY. In other words, use the combined knowledge you have so far gained to become familiar with your computer and MTX BASIC.

You will find that at first you will make a lot of mistakes. If you find that there are things happening which you cannot control then do not be afraid to RESET the computer and start again.

The AUTO command automatically places a new line number in the edit screen after you press <RET>. Try the example below to see how it operates:

Type AUTO 100,25. This will start at line number 100 and go up in units of 25.

When you have typed the last line of your program, or if you have made a mistake and wish to exit from AUTO, then press the CLS key, followed by <RET>. This will cancel the AUTO command. The CLS key will abandon the line you are working on whether you are working in AUTO or not. To return to AUTO, type AUTO followed by the next line number you need with a comma and step size as in the example above.

This can be a very useful and time saving command if you are simply copying a program already written, but can be a bit annoying if you are writing a program from scratch, as you will find you are forever jumping in and out of AUTO, wasting more time than you save!

The command AUTO has a second use, and that is for deleting sections of your program. When we were looking at ways of editing, it was mentioned that a line of program could be deleted by simply typing the line number followed by RETurn. Using the AUTO command it is possible to delete several lines quickly. First set the line number you wish to start deleting from. Then set the step size taking care to avoid lines which you still need. Press the RETurn key as each line number you want to delete appears. Used with caution, this can be an invaluable time-saving device.

See Reference Section AUTO, LIST, REM

CHAPTER 8.

USING DATA

OBJECTIVE: To design programs which can be used for handling information using the READ and DATA statements.

The commands we have used so far to do calculations, have involved placing numbers in locations and then executing the program. If we wished to use the same program again for a different set of figures there would have to be a fair amount of rewriting to change the data. In the apple sharing program for example all the LET statements would have to be changed if John had 4 apples, Kate 5 and Mark 6.

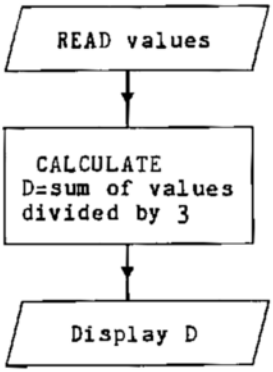
One way to solve this problem is to use the DATA statement. In this case the numbers are placed at the end of the program using a DATA instruction:

eg. 60 DATA 4,5,6

Notice that each number is separated by a comma but there is no comma included either after DATA or at the end of the line.

The command which tells the computer to place the DATA statement information into working memory is READ. Look at the flow chart and program below for the new sharing problem:

VARIABLES	
A	Marks Apples (4)
B	Johns Apples (5)
C	Kates Apples (6)
D	The number of apples each




```
10 REM NEW SHARING
20 READ A,B,C
30 LET D=(A+B+C)/3
40 PRINT D
50 STOP
60 DATA 4,5,6
```

In the program you can see that we have introduced a new command, STOP. The DATA statement in line 60 is not a command but holds data for the READ command in line 20. STOP in line 50 tells the computer not to execute the rest of the program.

This program can now be used for any sharing problem where there are three people. The only change which has to be made is to re-type line 60 with the new data. Before trying EXERCISE 11 run the program again using the following sets of data.

VARIABLES				
A	Marks Apples	(8)	(12)	(4)
B	Johns Apples	(8)	(10)	(11)
C	Kates Apples	(8)	(11)	(3)
D	The number of apples each			

Rewrite the program to enable you to share apples between five people and invent a number of apples for each person to place in the DATA statement.

Though the DATA statements can appear anywhere in the program, it is advisable to place them at the end, since it is then easier to add new lines here without disturbing your program.

Now try the problem in EXERCISE 11. The flow diagram has been partly completed for you.

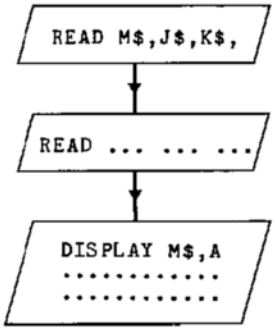
//

EXERCISE 11 PRODUCING A LIST OF EXAM RESULTS
In the ENGLISH RESULTS data table the results of three pupils are listed. Complete the program using the FLOW CHART, together with the READ, PRINT and DATA statements to print the list.

ENGLISH RESULTS	
Mark	64
John	68
Kate	45

VARIABLES TABLE	
Mark = M$	A = 64
John = J$	B = 68
Kate = K$	C = 45

FLOW CHART



PROGRAM

```
100 REM ENGLISH RESULTS
110
120 READ A,B,C
130 PRINT M$,A
140 PRINT J$,B
150
160 STOP
170 DATA Mark,John,Kate
180 DATA 64,68,45
```

Re-write the program to include more data:
Jill (72) Harry (48) George (56) Sandra (35)

//

See reference section DATA, READ, STOP

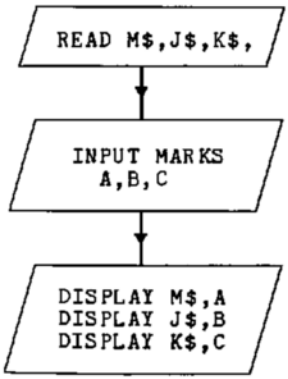
CHAPTER 9.

ENTERING DATA

OBJECTIVE: To design programs which allow the user to enter information whilst they are running.

The DATA statement is used to store data within a program before it is RUN. It is also possible to give data to a program whilst it is running. To do this you use the command INPUT. When the computer encounters an INPUT statement it waits to receive information typed in at the keyboard. The user inputs the information required followed by <RET>. The computer stores the information in the variables named in the statement. The ENGLISH RESULTS example from Exercise 8 would look like this if rewritten to use the INPUT statement.

FLOW CHART



PROGRAM

```
100 REM ENGLISH RESULTS 2
110 READ M$,J$,K$
120 INPUT "A =    ";A
130 INPUT "B =    ";B
140 INPUT "C =    ";C
150 PRINT M$,A
160 PRINT J$,B
170 PRINT K$,C
180 STOP
190 DATA Mark,John,Kate
```

If insufficient information is INPUT or non-numeric data is input to a numeric variable, a question mark will be printed after the entry and you must type the information again from the start.

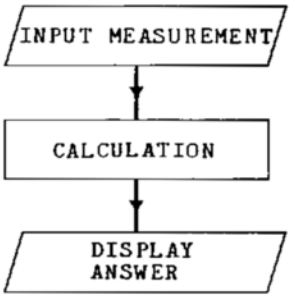
//

EXERCISE 12 INPUT
Redesign the rectangle program to find:

A)The area of a triangle.
B)The area of circle.
C)The circumference of a circle.

The area of a rectangle.

FLOW CHART



PROGRAM

```
100 REM AREA OF RECTANGLE
110 INPUT "LENGTH =     ";L
120 INPUT "BREADTH =    ";B
130 LET A = L * B
140 PRINT "AREA =     ",A
```

```
100 REM AREA OF A TRIANGLE
110
120
130
140
```

```
100 REM AREA OF A CIRCLE
110
120
130
```

```
100 REM CIRCUMFERENCE OF A CIRCLE
110
120
130
```

//

See Reference Section INPUT

CHAPTER 10

BRANCHING PROGRAMS (MAKING DECISIONS AND CONDITIONAL STATEMENTS)

OBJECTIVE: To design programs which instruct the computer to make decisions.

The programs you have been writing so far involve straight forward calculations where a result is obtained in a variable (LET A = 5 + 4, for example, where A is the variable) and then printed on the screen. In this Chapter we are going to look at the ways in which you can write programs that make decisions. To do this the computer considers whether a statement is true or false and depending on the logical conclusion it reaches may enter different calculation pathways. CONDITIONAL STATEMENTS such as IF and THEN are used: IF the answer is true a THEN statement would be used to tell the computer what to do next; and if false an ELSE statement could be used to tell the computer to take another calculation path. Think about this example in English;

IF the milk is fresh

THEN I will drink white coffee

ELSE I will drink my coffee black.

The use of IF THEN ELSE in MTX BASIC can be thought of in exactly the same way, the path taken by the program depends on whether the condition in the IF statement is true or not.

```
10 INPUT "YOUR AGE  ";A
20 IF A=0 THEN GOTO 10 ELSE GOTO 30
30 PRINT "YOUR AGE IS  ";A
```

The relation between A and 0 is a relation between two values.The example above is an operation to test the relationship of the two values, and is termed a BOOLEAN operation. There are six possible BOOLEAN operators which are listed in the table below.

OPERATOR	RELATION TESTED	EXPRESSION
=	is equal to	A = B
<>	is not equal to	A <> B
<	is less than	A < B
>	is greater than	A > B
<=	is less than or equal to	A <= B
>=	is greater than or equal to	A >= B

Where an expression includes relational and arithmetical operations the arithmetic is carried out first.

EXAMPLE

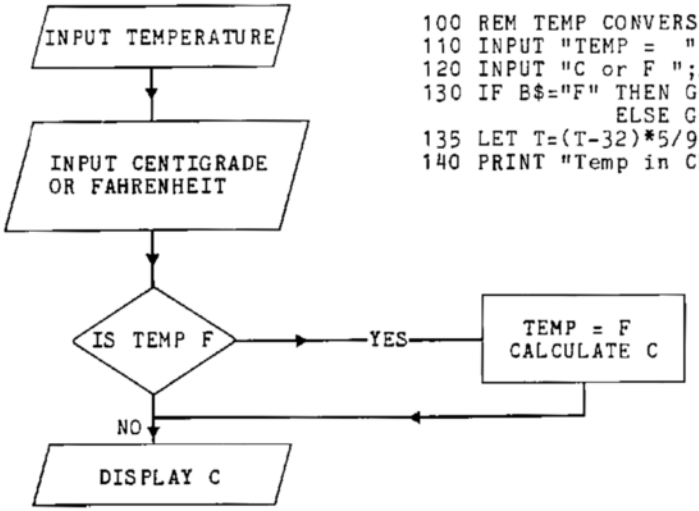
If we were to ask eight schools in the UK to send in the temperature recorded in their weather station at noon on a given day, it is likely that some schools would send their results in CENTIGRADE and some in FAHRENHEIT. If we wish to compare these temperatures we must convert them to either one scale or the other. The program below is an attempt to do this by converting the FAHRENHEIT results into CENTIGRADE.

DATA TABLE.

CENTIGRADE	
MANCHESTER	10
LIVERPOOL	11
BRIGHTON	13
CARDIFF	10

FAHRENHEIT	
SHEFFIELD	56
OXFORD	62
GLASGOW	52
NEWCASTLE	54

Look at the program below. Line 110 stores a temperature in variable T and line 120 stores F or C in B$ to identify the temperature as Fahrenheit or Centigrade. In line 130 the computer decides by asking if B$ is an F or C which branch of the program should be followed. IF the variable B$ holds an F then it has to be converted to centigrade and so the instruction 'THEN GOTO' is used to tell the computer to carry out the calculation in line 135. IF B$ holds a C 'ELSE GOTO' tells the computer to miss out line 135 and continue with the PRINT instruction in 140.




```
100 REM TEMP CONVERSION
110 INPUT "TEMP =  ";T
120 INPUT "C or F ";B$
130 IF B$="F" THEN GOTO 135
                ELSE GOTO 140
135 LET T=(T-32)*5/9
140 PRINT "Temp in CENTIGRADE=";T
```

COUNTERS

There are a number of ways to make this program more "user friendly". We may, for example, add a simple counter so that the computer knows how many temperatures it is going to work out. Counters are used where the same calculations are carried out on a lot of numbers. When you use a counter, your first step is to INITIALISE it (this is the number that you want the counter to count from). Usually counters are initialised to 0, but they can be set to any number you like. Then each time the calculation is performed 1 is added to the counter and the computer checks to see whether the desired number has been reached.

In the temperature conversion example the program could be rewritten like this:

```
10 INPUT "NUMBER OF TEMPERATURES ";N

20 LET C=0

30 INPUT "ENTER TEMPERATURE ";T
40 INPUT "IS IT FAHRENHEIT  ";B$
50 IF B$="N" THEN GOTO 70
60 LET T=(T-32)*5/9
70 PRINT T

80 LET C=C+1

90 IF C=N THEN GOTO 100 ELSE GOTO 30

100 PRINT "END"
```

Another way in which the program could be made more effective and easy to use is to redesign it to handle more than one entry at a time. Since we wish to produce a table giving a picture of the country as a whole we need to be able to INPUT all the data together and print the results in one table at the end.

One way to do this is to use the READ and DATA statements.

//

EXERCISE 13

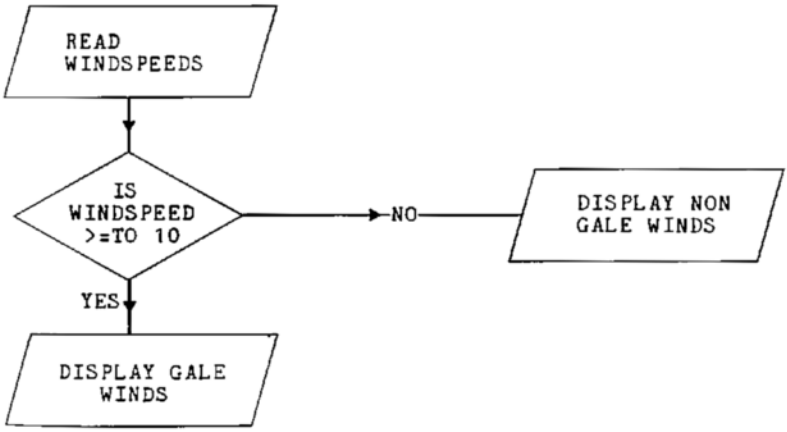
1) Redesign the temperature conversion program using the READ DATA statements to produce a single table.

2) Design a program to classify winds into gale force or non gale force.

VARIABLES TABLE.

WINDSPEED		
WINDSPEED 1	4	(A)
WINDSPEED 2	12	(B)
WINDSPEED 3	8	(C)
WINDSPEED 4	11	(D)
WINDSPEED 5	10	(E)
WINDSPEED 6	9	(F)

Speeds are on the Beaufort scale. A gale is any wind equal to or over force 10.



//

See Reference Section IF, THEN, ELSE, GOTO, BOOLEAN EXPRESSIONS

CHAPTER 11

PROGRAMS WITHIN PROGRAMS

OBJECTIVE: To introduce subroutines to carry out calculations whilst the program is running.

The GOTO statement is useful when used in simple branching programs. However when many calculations are used to solve a problem GOTO statements become complicated and difficult to follow. In order to keep your program organised and to make it run more efficiently it is often best to use subroutines.

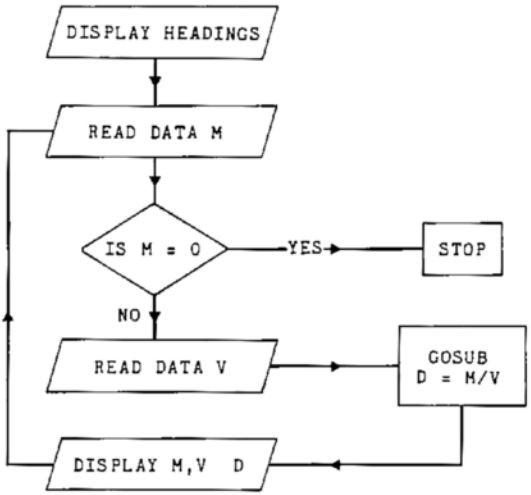
When the computer encounters a subroutine it leaves the main set of instructions, having recorded where it is up to, and then carries out a separate set of commands, returning on completion to the point of departure.

The commands used to tell the computer to leave and re-enter the main program are GOSUB (GO to the SUBroutine) and RETURN. You can see them used in the simple example below.

In this example you are given the values for the mass and volume of a set of liquids and the program is designed to work out the density of each and then print them all in a table.

```
100 REM DENSITY PROGRAM

110 PRINT "MASS","VOLUME",,
      "DENSITY"
120 READ M
130 IF M=0 THEN STOP
140 READ V
150 GOSUB 500
160 PRINT M,V,,D
170 GOTO 120
180 DATA 8,4,6,3,12,6,0
500 REM SUBROUTINE
510 LET D=M/V
520 RETURN
```



Notice that line 130 of the above program is used to instruct the computer to stop when a 0 value for M is encountered. At the end of the DATA line 180 you will find the 0. This is a common technique for terminating data.

Now try some more conversion programs.

//

EXERCISE 14 Design a program with a subroutine to solve the following:

1) To convert Pounds Sterling into Dollars at the rate $1.54 to the pound.

2) To convert pounds (weight) into kilos.

3) Think of other conversions which require a subroutine, and then design suitable programs.

Check your answers by converting known values

//

See Reference Section GOSUB, RETURN

CHAPTER 12

STRUCTURING YOUR PROGRAMS

OBJECTIVE: Using BASIC commands to structure programs with loops.

Until now, we have used GOTO statements to repeat calculations. Look at the following programs which both print out a list of numbers from 1 to 10.

```
10 LET I=1                      10 FOR I=1 TO 10 STEP 1
20 PRINT I                      20 PRINT I
30 LET I=I+1                    30 NEXT I
40 IF I>10 THEN STOP            40 STOP
50 GOTO 20
```

Loops are required so often in programs that a special command is provided to make them faster, more flexible and easier to understand.

The FOR statement tells the computer that it is at the start of a loop. This loop is called a FOR LOOP. Each FOR LOOP has a variable associated with it called its control variable. In the above example the control variable is I but could be any simple numeric variable.

```
10 FOR I ....
```

When the FOR LOOP is met for the first time a value is given to the control variable. In the example it is 1 but could be any number or mathematical expression.

```
10 FOR I=1 ...
```

The FOR LOOP is now executed until a NEXT statement is encountered with the same control variable. (Line 30 above).

The program now returns to the start of the FOR LOOP to see if it has finished. There are two more numbers in the FOR statement. The second is called the limit and the third is called the increment. The increment is now automatically added to the control variable and the computer tests to see whether the limit has been reached. In the example the limit is 10 and the increment is 1.

```
10 FOR I = 1 TO 10 STEP 1
```

If the limit has been reached, the program will jump to the statement following the NEXT statement. If not, the loop will be performed again with the new value of the control variable. The following program is a practical example.

```
100 PRINT  "G-MARK","FAHRENHEIT","CENTIGRADE"
110 FOR I=1 TO 8 STEP 1
120 LET F=250+I*25
130 LET C=(F-32)/9*5
140 PRINT I,F,,C
150 NEXT I
160 STOP
```

This program converts GAS MARKS used on domestic cookers to temperatures in both Centigrade and Fahrenheit and prints out a table. I is used to represent the GAS MARK. Line 110 tells the computer how the loop is to be operated. The computer understands that it is to begin with Gas Mark 1 and repeat the loop 8 times incrementing the GAS MARK by 1 step each time.
If the cooker for which the conversion is being made begins at .5 and increases in units of 0.5 to gas mark 8 then the line 110 would be rewritten to:

```
110 FOR I=.5 TO 8 STEP .5
```

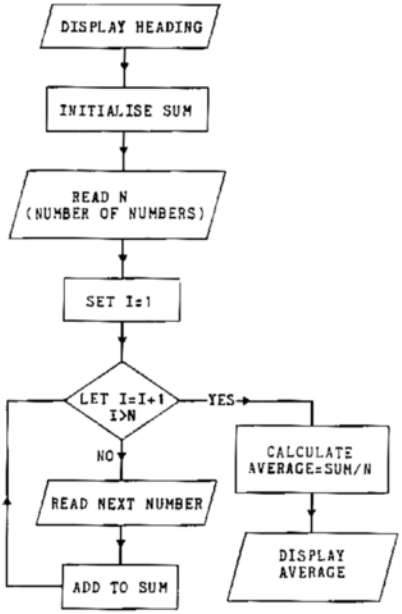
Line 150 is the end of the loop. The computer is told to return to the FOR statement and to carry out the instructions for the next value of I. It does this by increasing I by the value of STEP. If it has reached the limit the FOR NEXT loop is complete and control is passed to the next line in the program (ie line 160). The lines between the FOR and NEXT statement (120 - 140) are the instructions to be carried out for each step.

//

EXERCISE 15
Re-design your programs in exercise 14 (conversions) to use FOR, NEXT loops and to print out tables.

//

In the examples so far it has been simple to provide all the information for the FOR...NEXT statement. You may wish to write a program however, where the number of calculations constantly change. An example of this would be a program to estimate batting averages for a cricket team. In the example below, instead of specifying exactly the number of times the loop has to be carried out, a VARIABLE 'N' has been used, so that this information can be read while the program is running, i.e.:




```
100 REM BATTING AVERAGES
105 PRINT "BATTING AVERAGE";
110 REM SET SUM TO 0
120 LET S=0
130 REM READ VALUE OF N
140 READ N
150 REM SET UP LOOP
160 FOR I=1 TO N
170 REM READ SCORES
180 READ X
190 LET S=S+X
200 NEXT I
210 REM CALCULATE AVERAGE
220 LET A=S/N
230 PRINT A
240 DATA 7
250 DATA 125,0,45,67,83,90,68
```

In the previous example the FOR statement was applied to a situation where the number of times the loop is performed changes. Consider the program below which uses more than one FOR loop to automatically adjust tyre pressures on a car to 32.

```
10 INPUT "TYRE PRESSURE";P
20 IF P<>32 THEN GOTO 50
30 PRINT "TYRE PRESSURE CORRECT"
40 GOTO 10
50 IF P>32 THEN GOTO 100
60 FOR I=P TO 32 STEP 1
70 PRINT I
80 NEXT I
90 GOTO 30
100 FOR I=P TO 32 STEP -1
110 PRINT I
120 NEXT I
130 GOTO 30
```

See reference section FOR, NEXT, STEP

CHAPTER 13

MORE BRANCHING PROGRAMS (CONDITIONAL JUMPS)

OBJECTIVE: To design more complicated decision making programs

In the previous sections we have looked at how programs can be made to branch and loop. These methods are not particularly suited to the input of data which is continually changing. In a game of billiards between four players for example, as each player takes his turn his total is raised by his new score and the other scores are unaffected. The program below is a possible way to solve this problem.

```
10 REM ENTER PERSON AND SCORE
20 LET S1=0: LET S2=0: LET S3=0: LET S4=0
30 INPUT "PERSON NUMBER AND SCORE";P,S
40 IF S=5000 THEN GOTO 140
50 ON P GOTO 30,60,80,100,120
60 LET S1=S1+S
70 GOTO 30
80 LET S2=S2+S
90 GOTO 30
100 LET S3=S3+S
110 GOTO 30
120 LET S4=S4+S
130 GOTO 30
140 PRINT:PRINT
150 PRINT "PLAYER,,SCORE"
160 PRINT
170 PRINT "      1";S1
180 PRINT "      2";S2
190 PRINT "      3";S3
200 PRINT "      4";S4
210 PRINT:PRINT
220 PRINT "END OF GAME"
```

Notice that in line 40 the instruction to end the game is by the use of an IF statement which would only be true if an impossible situation arose, that is, a score of 5000 is input.

//

EXERCISE 16 ON..GOTO ON..GOSUB.

1) Design a game program for four players where the object of the game is for each player to guess a number between 1 and 100 which has been input by a fifth person. The structure of the program above will be a useful starting point.

//

See Reference Section ON GOTO

CHAPTER 14

MORE ABOUT VARIABLES

OBJECTIVE: To introduce the handling and manipulation of variables.

So far you have used data to calculate answers and then asked the computer to display the results on the screen. There may be times when you want to display the data as well. For example,if you were producing accounts you might need to list all the entries as well as totals and other calculations.

ARRAYS and DIM

You may remember from CHAPTER 2 that variables are locations where information is stored and that they are allocated letters to identify them. You may also remember that it is possible to extend the number of variables available by allocating a letter with a number. So, for example, the A variable may become A1,A2,....A9. If we place the numbers in brackets (A(1),A(2) and so on) the computer understands that the variables are linked together. In this case we have a set of numbers (A) which are sub-divided into members of the set by the use of the subscript (1,2,...9). This type of set is called an ARRAY.

In order that the computer can organise the storage of an ARRAY in appropriate locations it is necessary to tell the computer how many members of the set there will be. To do this you use the DIMension statement.

```
10 DIM S(40)
```

This example indicates that there is to be an ARRAY, which consists of 40 numbers. It is not essential for all forty variables to be used, you have told the computer that the number will not exceed 40. In other words the DIM statement places an upper limit on the size of the array to be set up. The example above therefore would tell the computer to set up the array as:

```
S(1),S(2),S(3),S(4).........S(40)
```

DIM statements can also be used to dimension strings. They are used to define the length of the string in much the same way as with numbers. For example, the DIM statement to make space for a string of length at most 14 characters would be:

```
10 DIM R$(14)
```

This would allow

```
20  LET R$="STEVEN JAMESON"
```

but not

```
20  LET R$="STEPHEN JAMESON"
```

If a DIM statement is not used before the first use of a string array, the string array is assumed to be of one dimension. In other words, space is made to hold a single string of characters. (see the reference manual for more details).

To make space for an array which can hold a number of strings we need to use a dimension statement to tell the computer how many strings and how long the longest one is going to be.

e.g.

```
10 DIM R$(20,10)
```

tells the computer that space is required for 20 strings of maximum length of 10 characters.

```
10 DIM R$(4,5)
20 DIM S(4)
30 FOR I=1 TO 4 STEP 1
40 READ R$(I),S(I)
50 PRINT R$(I),S(I)
60 NEXT I
70 STOP
80 DATA FRED,47,HARRY,62,JOE,26,SID,54
```

//

EXERCISE 17 DIM STATEMENTS

Re-design the above program to print the names of the 7 additional members of the cricket team, and at the same time to calculate the average age of the players

//

Try and work out what the following program is doing.

```
10 LET A$="123456789"
20 FOR I=1 TO 7
30 PRINT A$(I,3)
40 NEXT I
```

See the reference section DIM, 'MANIPULATING STRINGS'

CHAPTER 15

SORTING

OBJECTIVE: Using nested loops and arrays in programs designed to sort numbers.

One of the more useful tasks a computer can do for you is to sort large amounts of data into the order you require very quickly and efficiently. Sorting may be alphabetic, numeric (ascending or descending) or in fact almost any way you can think of.There are many different ways of sorting, but there is only space here to mention a few, and we will concentrate on one method in particular.

The main problem that all sorting techniques have to overcome is the large number of tests required. That is to say the computer is forced to ask many questions about the data in order to sort it effectively. Say, for example, we have three numbers which we will call A,B,and C.The first test would be to ask if A is greater than B If it is then A and B are in order. If A is less than B then the order has to be changed. To do this the number in B has to be switched to the variable A and A to B. A third location T is used to do this as below:

```
10 IF A<B
20 LET T=B
30 LET B=A
40 LET A=T
```

Having placed A and B in order we then have to place C in position. To do this there are two questions required, though we may get away with one.

They are:

1) is C greater than B,
2) is C greater than A.

By using this method as each new number is included in the sort, the number of questions to be asked increases. As you can imagine the number of questions needed to place the 50th number in order is daunting and would take a long time to compute.

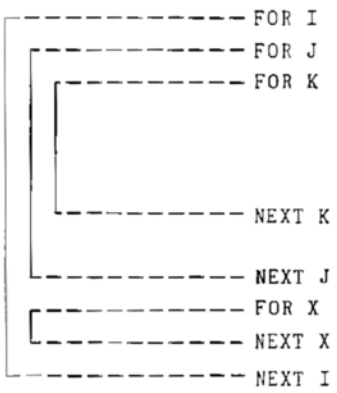
The RIPPLE SORT technique places the numbers into a row and is designed so that the computer asks only one question as it moves along the row: Is the number I am holding (which is the largest so far) larger than the next number? If the answer is yes then the computer moves on, if no then an exchange is made. By this method the largest number moves to the last position.

The process is then repeated by using a loop until all the numbers are placed in order. We have to tell the computer when to stop and this is achieved by setting a flag to zero. A flag is just a simple counter which can be looked at at any time to check the status of the program. The way flags are used in programs can be seen in the examples below. Each time an exchange is made then the flag 'F' in line 120 is set to 1. The main loop includes the line which sets the flag to 1 and so if at the end of the run it is still 0, no exchanges have been made and the sort is complete.

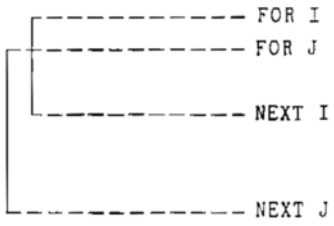
```
10 DIM A(20),B(20)
20 PRINT:PRINT
30 PRINT "NUMBERS","SORTED LIST"
40 READ N
50 FOR I=1 TO N STEP 1
60 READ A(I)
70 LET B(I)=A(I)
80 NEXT I
90 LET F=0
100 FOR I=1 TO N-1
110 IF B(I)<=B(I+1) THEN GOTO 160 ELSE GOTO 120
120 LET F=1
130 LET T=B(I)
140 LET B(I)=B(I+1)
150 LET B(I+1)=T
160 NEXT I
170 IF F=1 THEN GOTO 90
180 FOR I=1 TO N STEP 1
190 PRINT A(I),B(I)
200 NEXT I
210 DATA 10
220 DATA 6,4,8,1,3,2,5,7,9,10
```

In order to improve the efficiency of sorting tasks a number of different methods are used. One way is to put decision making FOR loops inside one another. This is called NESTING. Below are some combinations of permitted and illegal FOR...NEXT loops.

LEGAL LOOPS



ILLEGAL LOOPS



Work through these loops and see why they aren't allowed and,in fact would not work.

An example is given in the program below to place in order the Top Ten records based on sales.

```
10 DIM R$(40,40)
20 DIM S(40)
30 LET I=1
40 PRINT "RECORD NAME";
50 INPUT N$
60 IF N$="" THEN GOTO 150
70 LET R$(I)=N$
80 PRINT "SALES         ";
90 INPUT S(I)
100 LET I=I+1
110 GOTO 40
150 CLS
200 LET N=I-1
210 FOR I=1 TO N STEP 1
220 LET K=1
230 LET W$=R$(1)
240 LET MAX=S(1)
250 FOR J=1 TO N STEP 1
260 IF S(J)>MAX THEN LET MAX=S(J):
    LET K=J:LET W$=R$(J)
270 NEXT J
280 LET S(K)=0
290 PRINT W$,MAX
300 NEXT I
```

//

EXERCISE 18 DIM
1) Design a program to input football teams and their points and then sort them into a final order.

2) Design a program to sort the teams on the basis of points and goal difference

//

A useful way of sorting in larger programs is to separate the sort from the main program by placing it in a sub routine. Below is a listing of a useful sub routine doing exactly that. The instruction to call up the routine would be GOSUB 1000.

```
1000 REM SORTING SUB ROUTINE
1010 LET F=0
1020 FOR I=1 TO N-1
1030 IF A(I)<= A(I+1) THEN GOTO 1080
1040 LET F=1
1050 LET T=A(I)
1060 LET A(I)=A(I+1)
1070 LET A(I+1)=T
1080 NEXT I
1090 IF F=1 THEN GOTO 1010
1100 RETURN
```

REMEMBER THAT THIS IS A SUBROUTINE

CHAPTER 16.

MULTI DIMENSIONAL ARRAYS

OBJECTIVE: To analyse data in tabular form.

In CHAPTER 14 the DIM statement was used to define arrays. We are now going to look at how an array can be used to store two dimensional data. As before, the DIM statement is used. Consider the following section of a program.

TWO DIMENSIONAL ARRAYS

```
10 DIM X(5,4)
```

This statement sets up an array which has five rows and four columns. This looks like;

X(1,1)	X(1,2)	X(1,3)	X(1,4)
X(2,1)	X(2,2)	X(2,3)	X(2,4)
X(3,1)	X(3,2)	X(3,3)	X(3,4)
X(4,1)	X(4,2)	X(4,3)	X(4,4)
X(5,1)	X(5,2)	X(5,3)	X(5,4)

And could store the information:

	X	2X	X^2	X^3
X=1	1	2	1	1
X=2	2	4	4	8
X=3	3	6	9	27
X=4	...	... etc	...	...
X=5	...	...	...	...

//

EXERCISE 19 Two dimensional arrays

1) Design a program to complete and print the table above.

2) In the example below part of the third year exam results are given. All the figures are percentages and each class took the same exam in each subject. In order to look at the progress of children in the three subjects a program has to be devised to work out:

a) The average results for each class in each subject.
b) The average result for the year in each subject.
c) To print a table of results as below
d) To print a list in order of score for each subject.

YEAR 3

CLASS	PUPIL NUMBER	GEOGRAPHY	HISTORY	MATHEMATICS
3A	1	58	62	23
	2	45	58	29
	3	67	76	53
	4	53	45	12
	5	68	72	43
3B	25	46	43	51
	26	48	41	45
	27	53	41	55
	28	49	36	62
	29	51	68	65
3C	55	43	38	42
	56	54	37	51
	57	47	56	49
	58	43	43	28
	59	54	45	43

The input part of the program and a section of program which points to a possible way to complete the design has been completed for you. Notice that the array is initialised by the use of input statements making the program adaptable to many different situations.

```
10 REM INPUT PROGRAM
20 INPUT "MAX NUMBER OF PUPILS PER CLASS";PUPIL
30 INPUT "NUMBER OF CLASSES";CLASS
40 INPUT "NUMBER OF SUBJECTS";SUBJECT
50 DIM R(PUPIL,CLASS,SUBJECT)
60 REM START ENTERING DATA
70 INPUT "PUPIL,CLASS,SUBJECT,MARK";P,C,S,M
80 IF P=0 THEN GOTO 110
90 LET R(P,C,S)=M
100 GOTO 70
110 REM CONTINUE
```

//

```
1000 REM AVERAGE PROGRAM
1010 REM ASSUME RESULTS ARE IN R
1020 FOR C=1 TO CLASS STEP 1
1030 LET T=0
1040 LET N=0
1050 FOR P=1 TO PUPIL STEP 1
1060 FOR S=1 TO SUBJECT STEP 1
1070 LET M=R(P,C,S)
1080 IF R(P,C,S)<>0 THEN LET T=T+M:LET N=N+1
1090 NEXT S
1100 NEXT P
1110 LET A=T/N
1120 PRINT "AVERAGE OF CLASS ";C;" IS ";A
1130 NEXT C
```

In this example the same data is used for a number of calculations. MTX BASIC provides a command for placing data back into the working memory for further analysis. The command RESTORE is used with the READ and DATA statements to do this as follows;

```
10 READ N
20 LET T=0
30 FOR S=1 TO N STEP 1
40 READ X
60 LET T=T+X
70 NEXT S
80 LET A=T/N
100 PRINT:PRINT
110 PRINT "AVERAGE SCORE IS    ";A
120 RESTORE 0
125 PRINT
126 PRINT "PUPIL   MARKS      DEVIATION"
127 PRINT
130 READ N
140 FOR I= 1 TO N STEP 1
150 READ Y
160 PRINT I,Y,Y-A
170 NEXT I
180 STOP
190 DATA 5
200 DATA 45,67,89,34,51
```

See the reference section RESTORE

CHAPTER 17

FORMATTING WITH PRINT

OBJECTIVE: To introduce methods of formatting when using the PRINT command.

We have used the PRINT command to print tables by using commas to take you to the next TAB position. You may however wish to set up a table or enter text which does not start at the first TAB position and then use each subsequent position. MTX BASIC has a command CSR (CURSOR) which is used with the PRINT command to help you format your work in this way.

The first step is to set the starting position by giving the CURSOR two co-ordinates. The first is the number of columns across the top of the screen and the second the number of rows down.

```
CSR 3,4
```

will place the cursor three character spaces from the left hand edge of the screen and four lines down from the top. (A fuller explanation is found in GRAPHICS.)

Try this simple example to print your name in the middle of the screen.

```
10 REM YOURNAME
20 CSR 12,15:PRINT "YOURNAME"
```

Notice we have placed the PRINT command on the same line as the CSR command. This is all right if you separate the commands with a colon. You can use multiple statement lines in this way provided you do not exceed the length of the EDIT screen.

See Reference section CSR

CHAPTER 18

MATHEMATICAL FUNCTIONS

OBJECTIVE: To introduce arithmetical functions and their use in programs.

Remember in chapter 3 we looked at roots and wrote the square root of four as 4^(1/2). At the time it was hinted that there were more efficient ways of doing these calculations, and if you haven't discovered these for yourself already, then now is the time. An example of the method you have used to calculate square roots is below on the left with the alternative on the right:

```
10 LET X=16                    10 LET X=16
20 LET Y=(X)^0.5               20 LET Y=SQR(X)
30 PRINT Y                     30 PRINT Y
```

A complete list of mathematical functions is included in the reference manual. It is worth having a look at all of these now, because even if you have discovered some of them, it is likely that there are some you have not, and it may save you time in the future if you know what is available.

//

EXERCISE 21 FUNCTIONS

1) The following program prints out the values of SIN and COS for values between .1 and .9. Alter it to print values of other functions.

```
10 FOR X=.1 TO .9 STEP .1
20 PRINT " X",,"SIN(X)",,"COS(X)"
30 PRINT
40 PRINT X,SIN(X),COS(X)
50 NEXT X
```

//

See Reference Section SIN, COS, PI and find the other functions

CHAPTER 19

STRING FUNCTIONS

OBJECTIVE: The use of String functions as controls.

There are two types of string functions. The first type is used to instruct the computer to perform operations rather like program statements. The second is used to manipulate strings.

We will first deal with string functions as operators. The keyboard can be read by the use of the function INKEY$. This is in practice a very useful function since it enables you to write programs where the computer interacts with the person operating the keyboard. The short program below is an example of this where the computer expects the operator to answer "Y" and any other response will result in a loop.

```
10 PRINT "PRESS Y TO CONTINUE"
20 LET A$=INKEY$
30 IF A$<>"Y" THEN GOTO 20
40 PRINT "YOU PRESSED Y"
```

If you cannot understand this program insert

```
25 PRINT A$;
```

Another useful string function is CHR$. This function is used to send character codes to the screen, usually because the codes don't have a corresponding printable character.

e.g.PRINT CHR$(65)

This will print a capital A at the next position because 65 is the ASCII code of 'A'. The Appendix includes a full list of the character codes.

In the same way, printing can be controlled by using the special control characters .

e.g.

```
PRINT CHR$(12)  CLEARS THE SCREEN.
PRINT CHR$(26)  MOVES CURSOR TO HOME
PRINT CHR$(10)  MOVE CURSOR DOWN
```

The second type of string function is concerned with the manipulation of strings.

If you want to display only part of a string for example,MID$ can be used. Look at the following program:

```
10 LET A$="ABCDEFG"
20 PRINT MID$(A$,3,2)
```

When this is run CD will appear on the screen. In the instruction you have told the computer to go to the third letter which is `C' and print two letters.

LEFT$ and RIGHT$ are used to instruct the computer to count from the beginning of the string (LEFT$), and the end of the string (RIGHT$). The number of characters specified will be printed. Replace line 20 with the following lines.

```
20 PRINT LEFT$(A$,3)          ABC will appear.
20 PRINT RIGHT$(A$,3)         EFG will appear.
```

See Reference Section LEFT$, RIGHT$, MID$, CHR$ and other string functions

CHAPTER 20

SIMPLE GAMES AND RANDOM NUMBERS

OBJECTIVE: To introduce the methods used in the design of games programs.

Most of the games constructed for the computer involve complicated graphics which are controlled by a series of loops, conditional jumps and sub-routines. In this CHAPTER we are going to concentrate on the programs rather than the graphics. Later on when you feel more confident you could attempt to make the games you create here more interesting by adding appropriate graphics but don't worry about that at the moment.

An important feature of any game is that the events in it occur by chance and are in no way predictable. To produce this 'random' effect BASIC has a function RND and a command RAND. Random numbers are created by the random number generator. To activate this you need to set a starting point in the RAND statement and set a limit of numbers to be generated in the RND statement. We have included a program designed to fill out a pools coupon with numbers derived by the computer at random. In this case the coupon allows you to enter 20 possible draws and therefore, you need to generate 20 numbers.

```
10 RAND 5
20 FOR I=1 TO 20 STEP 1
30 PRINT INT(RND*64+1),
40 NEXT I
50 PAUSE 5000
```

When you have tried this program a few times you will see that it has a problem. It always starts from the same point and therefore, it continues to produce sets of identical numbers. This is called a pseudo-random number sequence where RAND 5 will always produce the same set of numbers. RAND 6 would produce a new set of numbers but then repeat them each time it is used. This is true for all positive whole numbers used in the RAND statement.

To obtain truly random numbers you would use a negative value in the RAND statement. Now try RAND -5 in line 10. As with positive values you can use any whole number.

RND returns random numbers in the range 0 to .9999999.
To obtain a different range, the result can be multiplied by a scaling factor.

Line 30 tells the computer that you wish to display a whole number (INT) between 1 and 64. The computer would start from 0 and go to 63 if no further instruction is given. The easiest way to solve this is to simply add 1 to each number generated and so:

```
INT(RND*64+1)
```

//

EXERCISE 22 RANDOM NUMBERS

1) Design a program to generate a random number table in 10 columns from 1 to 99.

2) Complete the dice throwing program below.

```
10 RAND 5000
20 LET D1=INT(RND*6+1)
30 LET D2=INT(RND*6+1)
40 PRINT:PRINT
50 PRINT "DICE 1=    ";D1
60 PRINT "DICE 2=    ";D2
70 IF D1=D2 THEN GOTO 80 ELSE GOTO 100
80 PRINT "PRESS SPACE FOR EXTRA THROW"
85 LET A$=INKEY$
86 IF A$<>" " THEN GOTO 85
90 GOTO 20
100 INPUT "NEXT TURN ? Y FOR YES N FOR NO";A$
110 IF A$="Y" THEN GOTO  20 ELSE GOTO 120
120 PRINT "GAME OVER"
```

3) Redesign the above program to allow four dice to be used by three or four players and include a counter to print scores for each player in a best out of three game.

//

There are two programs below to set up a game of bingo. By making these into subroutines and by properly formating the cards in an array you could enable the game to be run on the screen. Try first of all with two players.

BINGO

```
10 REM BINGO NUMBERS
20 DIM A(99)
30 CLS
40 PRINT "PRESS A KEY FOR NEXT NUMBER"
50 INPUT A$
60 LET X=INT(99*RND+1)
70 IF A(X)=1 THEN GOTO 60
80 LET A(X)=1
90 PRINT X
100 GOTO 40
```

BINGO CARD

```
10 REM BINGO CARD
20 DIM B(30)
30 DIM R(100)
40 CLS
50 FOR I=1 TO 15
60 LET X=INT(RND*99+1)
70 IF R(X)=1 THEN GOTO 60
80 LET R(X)=1
90 LET N=INT(RND*30+1)
100 IF B(N)<>0 THEN GOTO 80
110 LET B(N)=X
120 NEXT I
130 FOR I=1 TO 10
140 FOR J=1 TO 3
150 PRINT B((J-1)*10+I),
160 NEXT J
170 PRINT
180 NEXT I
```

See Reference section RAND, RND, INT

CHAPTER 21

MATRICES

OBJECTIVE: To introduce matrix operations in basic.

The two dimensional arrays introduced in CHAPTER 14 were forms of MATRICES. The principal difference between arrays and matrices is the way in which arrays are dealt with. In the case of two dimensional arrays we are concerned with the manipulation of separate parts of the arrays. The Geography results for 3B for example. In a matrix operation we are concerned with the whole matrix, so any operations affect the matrix as a whole.

There are a number of useful operations which can be carried out in this way on tables of information. Matrices can be added to each other, multiplied, divided, and constants can be applied to update them. Monthly sales figures, for example, can be combined to produce quarterly or yearly totals.

The sub-routines below can be used to set up, input data and print out matrices. They would be used as normal GOSUB/RETURN routines within a program. We have used a 3x3 matrix here but clearly by changing the I,J values you can design matrices of any dimension.

```
SUBROUTINE TO PRINT OUT A MATRIX
1000 FOR I=1 TO 3
1010 FOR J=1 TO 3
1020 PRINT A(I,J)
1030 NEXT J
1040 NEXT I
1050 RETURN

SUBROUTINE  TO SET A TO 0
2000 FOR I=1 TO 3
2010 FOR J=1 TO 3
2020 LET A(I,J)=0
2030 NEXT J
2040 NEXT I
2050 RETURN

SUBROUTINE TO INPUT DATA
3000 INPUT "I,J,DATA ";I,J,D
3010 IF D=99999 THEN RETURN
3020 LET A(I,J)=D
3030 GOTO 3000
```

This method can be used to add,copy, multiply and apply constants to the matrix. For example in the first sub routine above the replacement of line 1020 with:

```
1020 LET A(I,J)= B(I,J)
```

will lead to matrix B being copied into matrix A.

```
1020 LET C(I,J)= A(I,J)+B(I,J)
```

Now matrix A is added to B and the sum is copied into C.

```
1020 LET A(I,J)= 5*A(I,J)
```

All the elements of matrix A are multiplied by the constant 5.

To multiply two matrices a slightly more complex routine is required to fit the rules of matrix algebra. Consider the example below where matrix A has R rows and C columns and matrix B has C rows and S columns.

```
4000 REM SUBROUTINE TO MULTIPLY MATRICES
4010 FOR I=1 TO R STEP 1
4020 FOR J=1 TO C STEP 1
4030 LET D(I,J)=0
4040 FOR K=1 TO C STEP 1
4050 LET D(I,J)=D(I,J)+A(I,K)*B(K,J)
4060 NEXT K
4070 NEXT J
4080 NEXT I
4090 RETURN
```

//

EXERCISE 23
Mr Jones has three fish and chip shops and the table below shows the numbers of fish sold in the four quarters of the year.

		Cod	Haddock	Plaice
Shop 1	JAN - MAR	3,456	460	212
	APR - JUN	2,458	238	146
	JUL - SEP	1,845	67	35
	OCT - DEC	4,153	354	286
Shop 2	JAN - MAR	2,998	342	189
	APR - JUN	2,135	154	89
	JUL - SEP	1,225	52	38
	OCT - DEC	2,509	189	139
Shop 3	JAN - MAR	4,806	589	354
	APR - JUN	2,678	453	302
	JUL - SEP	2,688	220	148
	OCT - DEC	4,766	554	386

Use the sub routines to set up matrices to print out the data in the form above. It is possible to add matrices together and therefore, it is possible to work out the following totals:

1) The 6 monthly returns for each shop.
2) The yearly returns for each shop.
3) The combined figures for all shops.
4) Create a table to express each shops monthly figure as a percentage of the total sales for all shops.

//

PART 2

NODDY

In the early chapters you experienced how difficult it is to format your work when writing programs in BASIC. The new language NODDY has been designed to simplify text handling. The second important advantage of NODDY is that it allows you, with very little programming knowledge to write your own interactive programs. As you would expect the method of writing programs involves planning in advance and understanding the commands. However, since there are only ELEVEN commands this is not a difficult language to master.

NODDY COMMANDS

B	BRANCH	E	ENTER	P	PAUSE
I	IF	A	ADVANCE	L	LIST
G	GOTO	R	RETURN	O	OFF
		S	STACK	D	DISPLAY

The use and meaning of the commands will become obvious as we work through some examples of NODDY programs.

NODDY is accessed through BASIC by typing NODDY. You will see later that this is not accidental but to enable you to write more complicated programs where NODDY and BASIC work together.

Type NODDY <RET>

Noddy will appear at the bottom of the screen.

Now type **NAME**

Noddy>NAME (make sure that you type this in capitals.)

Press the <RET> key.

NAME will appear at the top of the screen.

This is the title of the page.

Move the cursor using the EDIT keys and type some information about yourself. Remember that each NODDY page is treated by the computer as a separate entry and so you do not press <RET> until you have typed all the information you wish to store. If you make a mistake and <RET> before you have finished simply type NAME <RET> again and the page will be placed back on the screen.

DO NOT PRESS THE CLS KEY SINCE THIS DELETES THE PAGE.

Continue as before and when the page is complete press <RET>.

You have just created a NODDY page called NAME.

If you now type DIR <RET>, the screen will be cleared and NAME will appear in the top left corner. This is the NODDY directory and tells you what pages are present. In this case the page title NAME will appear. When you type DIR make sure that you are using capitals since if you use dir you will create a page called 'dir'.

Now type NAME again, the information you entered before will be printed on the screen just as you typed it. If you want to change or add information edit the screen using the cursor keys and when you have finished <RET>.

Remember that the CLS key is used to remove a page from the system. When you press the key the page currently on the screen is lost. Though this key is very useful for editing out redundant pages it can be frustrating to watch an hours work disappear in a moment of carelessness.

NODDY provides you with a means of storing and displaying textual information.

When you typed NAME <RET> the first time, you were telling the computer that you wanted a page of text which you could refer to by the title "NAME".

Try creating other pages with different titles. Each time you create a page and press <RET>, Noddy should appear at the bottom of the screen.

When Noddy appears you can:

1) enter another page by typing a new title

2) type DIR to see what you have done.

3) look at a page already in the DIRectory by typing the title.

4) return to BASIC by pressing the CLS key followed by <RET>. This is one of the occasions where it is safe to use <CLS>.

To make sure that you are in the correct mode to CLS it is useful to get into the habit of typing DIR before returning to BASIC. This gives you an opportunity to check that all the files you require are present and avoids the situation where work is lost.

To return to BASIC from a NODDY page:

Type DIR <RET>

Press the <CLS> key followed by <RET>.

Ready will appear at the bottom of the screen.

If you return to BASIC you will not lose your work provided you do not switch the computer off. When you return to NODDY the pages will be just as you left them. You may wish to make a more permanent record.

To save NODDY files use the system described in chapter 1 on saving and loading programs. The NODDY file is given a name as with a BASIC program. If both NODDY and BASIC programs are present at the same time they will both be saved together. Similarly if you erase a BASIC program by typing NEW then the NODDY pages will go as well. You should think of NODDY and BASIC as languages linked closely together.

To write programs in NODDY special program pages are set up using the commands described above. To show you how each of these commands work we have set up four programs to store telephone numbers each one capable of better storage and recall than the last.

The first program consists of a telephone page and a program page. We shall call the telephone page FRED and the program page PROG1.

Type NODDY <RET>

Type FRED <RET>
Fred's telephone number is 555686

(Enter this page by pressing the return key.)

The program page uses three commands DISPLAY,PAUSE and RETURN. Each command is preceded by * to tell the computer to regard the next entry as a command. Now type the following page called PROG1.

PROG1

```
*DISPLAY FRED.
*PAUSE    *PAUSE
*RETURN
     <RET>
```

The first line *DISPLAY FRED. tells the computer to place the page called FRED on the screen. Where a page title is referred to in this way the page name is completed by a full stop. If you forget to do this the computer will not be able to carry out the search.

The second line tells the computer to keep the information on the screen for the length of two PAUSES (approximately 1 second per pause)

The third line uses the command RETURN to return you to BASIC after the PAUSES are complete.

Type DIR to see the page names.

To run this program you first of all have to go into BASIC. To do this press the CLS key followed by <RET> and Ready will replace Noddy at the foot of the screen. The word used to run a NODDY program is PLOD. This should be followed by the name of the program page in inverted commas.

Type PLOD "PROG1" <RET>

If you wish to run a NODDY program a number of times then it is best to place the PLOD instruction in a program line.

```
10 PLOD "PROG1"
```

Each time you wish to run the program simply type RUN <RET> and the NODDY program will be activated.

The second program page (PROG2) will allow you to RETURN to BASIC by pressing one of the keys. The command used to achieve this is *ENTER which is not dissimilar to the BASIC input command. Where *ENTER is used the computer waits for a key or keys to be pressed before continuing with the program. Type in and run the program as before calling the page PROG2. When the page FRED appears press any key followed by <RET> or just <RET>, the program will continue and Ready will appear at the bottom of the screen as you return to BASIC.

PROG2

```
*DISPLAY FRED.
*ENTER
*RETURN
```

Program 3 uses the commands *IF, *GOTO and uses labels to place you more in control of the program. *IF is used to instruct the computer to ask if your *ENTER is the correct one. If it is then the computer will move onto a different part of the program as required. This is carried out by using a label. For example in line 3 the instruction *IF R,r tells the computer to compare the *ENTER with R and if R is the key pressed to find a letter `r' and continue from there. So that the computer does not confuse the 'r' at the beginning of the new program line with any other it looks for an 'r' preceded by ^.

(Labels can be any character on the keyboard and you should attempt to work out the best system for you to use. The important thing is to be consistent and keep to a plan.)

If a key other than 'R' is pressed then the program continues with *GOTO PROG3. The *GOTO command is used in this page to return the control back to the beginning of the program page we are in. Normally *GOTO would be used to activate other program pages. Notice the fullstop after PROG3 and the position of ^r.

PROG3

```
        *DISPLAY FRED.
        *ENTER
        *IF R,r
        *GOTO PROG3.
    ^r *RETURN
```

BRANCH

A better way to take control to the beginning of the present Program page is to use the command *BRANCH. PROG4 illustrates the use of *BRANCH and extends the use of labels to allow you to use

a *ENTER to print FRED on the screen.

PROG4

```
    ^t      *ENTER
            *IF F,a
            *IF R,r
            *BRANCH t

    ^a      *DISPLAY FRED. *BRANCH t
    ^r      *RETURN
```

The final program in this series PROG5 allows us to use the pages as a telephone directory. The first step is to create more pages.

Type SID <RET>

Sid's telephone number is 555987

BERT

Bert's telephone number is 555321

Now type in the program page

PROG5

```
 ^t         *ENTER
            *IF F,a
            *IF B,b
            *IF S,c
            *IF RET,r
            *BRANCH t

     ^a     *DISPLAY FRED.        *BRANCH t
     ^b     *DISPLAY BERT.        *BRANCH t
     ^c     *DISPLAY SID.         *BRANCH t
 ^r         *RETURN
```

The first six lines of the program are a loop where the computer is waiting for a *ENTER of F,B,RET or S. If any other input is received then you will *BRANCH to t. If RET is entered then you will branch to r and RETURN to BASIC.
If F,B or S are pressed with a <RET> then control is passed to labels a,b and c respectively. As each is displayed the program branches to ^t at the beginning of the program and you are ready to begin the process again.

//

EXERCISE 24 NODDY
Improve the final address book program by getting the computer to display a MENU page at the beginning of the program and arranging the program so that it returns you to the MENU after each display. The MENU page is designed for you. Your task is to amend the page PROG5.

MENU

There are three telephone numbers in the directory:

SID ,FRED, BERT.
To display their numbers type the first letter of their names and press the <RET> key.
To return to BASIC type RET

As a further exercise try to design a program to hold your own address book.

//

To further illustrate NODDY programs, the next example shows you step by step how to create a program to simulate a book.

When reading a book there are a number of mechanical tasks required. You need to be able to turn a page, look through chapters to find your place, look back to check on some detail or if you wish to cheat look at the last page to find out "who dun it!".

To write such a program in basic would involve a fairly complicated program with many loops perhaps using subroutines and so on to enable you to call up the required pages. Then there would be the difficult task of formatting each page. NODDY requires only one program page, a contents page and a contents page for each chapter. The plan for the program is shown below.

STAGE 1
WHICH CHAPTER

CHAPTER 1	CHAPTER 2	CHAPTER 3
PETS	FARMS	ZOO

STAGE 2
WHICH PAGE

P1 DOGS	P1 SHEEP	P1 LIONS
P2 CATS	P2 PIGS	P2 ZEBRA
P3 MICE	P3 CATTLE	P3 SNAKES

There are no hard and fast rules as to how you approach the task of setting up the book. It is often easier to start with contents and work through the book, rather than begin with the program since you may wish to change the contents.

The book you are about to write is called MAMMALS.It consists of four chapters as in the plan and page one is entitled DOGS.

Type DOGS <RET>

Type on the remainder of the page, in any form you wish, some information about dogs. When you have completed the page press the <RET> key and the page will be saved exactly as you typed it. Repeat this for each page in the plan.

The next stage is to write a contents page for each chapter.

As before you first type a title to the page followed by the information required as below.

```
CHAP1                PETS

               CONTENTS
Choose which page and type P1,P2, or P3
                     P1 DOGS
                     P2 CATS
                     P3 MICE
```

Complete a contents page for each chapter. To check whether all the information you need has been input type DIR and a directory of your pages will appear on the screen as below.

```
CHAP1        CHAP2        CHAP3
DOGS         SHEEP        LIONS      (NB The order of the directory
CATS         PIGS         ZEBRA       varies according to the inputting
MICE         CATTLE       SNAKES
```

In order to start in the book at the appropriate chapter a Chapter contents page is needed. So as not to confuse this page with those contents pages already input we will call this page TITLES.

Type TITLES <RET>

Choose which chapter and type 1,2,3 or RETURN to BASIC.

```
1 CHAPTER 1 PETS
2 CHAPTER 2 FARM
3 CHAPTER 3 ZOO
R RETURN
```

This completes the contents of the book and all that remains is to write the program page. As before we give this page a title (MAMMALS) and then type in a series of command statements. You will notice that the full form of the commands has been replaced by the use of a single letter. Also notice that *E can appear on the same line as *D and that the *IFs are all grouped together. The labels indicated by ^ preceding a letter have been structured in such a way that it is clear which letters refer to title pages and which to pages of text. Your NODDY programs will work without all this careful formatting. However, when you arrange your programs in this way you reduce the number of mistakes you are likely to make.

Examine the program carefully line by line as you input, thinking about what is the function of the line and how does it carry out the desired instructions:

Line 1 displays the main contents page called TITLES and tells the computer to wait for an input to be *ENTERED from the keyboard.

Line 2 compares your input with the expected 1,2,3 or R for RETURN and instructs the computer to find labels a,b,c or r respectively and continue from the label. If any other letter or number is input then control is returned by the *BRANCH t command to the beginning of the program.

Continue analysing the program in this way.

```
MAMMALS
 ^t        *D TITLES. *E
           *I 1,a *I 2,b *I 3,c *I R,r *B t

 ^a        *D CHAP1. *E
           *I P1,g *I P2,h *I P3,i *B t
    ^g     *D DOGS.       *B d
    ^h     *D CATS.       *B d
    ^i     *D MICE.       *B d
 ^b        *D CHAP2.  *E
           *I P1,j *I P2,k *I P3,l *B t
    ^j     *D SHEEP.      *B d
    ^k     *D PIGS.       *B d
    ^l     *D CATTLE.     *B d
  ^c       *D CHAP3.   *E
           *I P1,m *I P2,n *I P3,o *B t
    ^m     *D LIONS.      *B d
    ^n     *D ZEBRA.      *B d
    ^o     *D SNAKES.     *B d
  ^d       *E *B t
  ^r       *R
```

Press the enter key to save the program page "MAMMALS" and provided you have made no errors inputting the program it is ready to run. If you have made a mistake the computer will give you one of three error messages: 'NO DATA ERROR' ,'OVERFLOW' or 'MISSING SYMBOL' and return you to BASIC.

No data means that the computer is looking for a page title that it cannot find. It is more likely that you have not entered the page, however it could be that your entry for the page is misspelt or that you have forgotten a space etc.

Overflow occurs where the computer has reached the end of the program page whilst looking for a label or command.

Missing Symbols could occur if * or . are missing from the program page.

These error messages are only hints about the error. To find the error, the message should be considered together with the page on which the error occured.

To correct the page press the CLS key and type NODDY. When Noddy appears on the screen, type the page title MAMMALS and the page is ready for editing. Check that all punctuation is correct and when you are satisfied that it is correct press the enter key again.

Go back to BASIC.

Use the command PLOD to run the program.
Clearly this program can only operate a book with three chapters each with three pages. It would be a very limited book and without radical alteration cannot be increased in size. The program barely fits on the page. To get around this problem the *GOTO statement allows you to switch to another program page. A better design would be to make each chapter contents a new program page and instead of using the *B t you would use *GOTO MAMMALS. for example.

//

Exercise 25 NODDY Book Program
Redesign the program to contain four chapters each with four pages.

//

The *GOTO statement is one of four program handling commands.

Program pages are stored in a stack rather like a stack of plates. When they are used they are taken from the stack from the top. Imagine you have three program pages to run one after the other. Using the STACK statement it would be written like this:

*STACK PROG3,PROG2,PROG1.

They would be taken from the stack in the order PROG1,PROG2 and finally PROG3.

The command *STACK is used to tell the computer the order of programs to be run. They are taken from the top of the stack each time and therefore in the example above, PROG1 would be run first,PROG2 second and PROG3 third.

The command *ADVANCE tells the computer to advance or move through the program stack. ie remove and execute the program on top of the stack.

The *OFFSTACK command tells the computer to take the next program off the stack without executing it.

We have designed a set of programs to show you how these commands operate.

Set up the following three pages called AA,BB and CC.

AA

AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA

BB

BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB

CC

CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

The main program page is called PROG and looks like this:

PROG

```
     *S  PROG,PA,PB,PC.
     *A
```

There are three other program pages called PA,PB, and PC.

PA

```
     *D AA. *P *P *A
```

PB

```
     *D BB. *P *P *A
```

PC

```
     *D CC. *P *P *A
```

When you PLOD "PROG" the computer pushes four program pages onto the stack:

```
PC
PB
PA
PROG
```

*A at the end of PROG takes the first program of the stack which is PC and executes it. CC is displayed for the length of the pauses *P. Meanwhile PC has been discarded and now the stack looks like this:

PB
PA
PROG

*A at the end of PC instructs the computer to take PB from the stack into working memory execute and discard it. This process is continued until PROG is reached and the stack is reassembled.In this way a loop has been formed.

To stop the program press the BRK key. If you insert an OFFSTACK COMMAND in program PC as below then PB is taken offstack each time and missed out.

```
PC
     *D  CC. *P *P *O *A
```

//

Exercise 26 Noddy Program Handling Commands
Use the principals in this section to design a program to enable you to scan the four chapter contents pages in Exercise 1.
A further advantage would be to use the *OFFSTACK command to create an option where you only scan the chapters beyond the point in the book you have reached.

//

The final command which we have not used as yet is *LIST. You cannot use LLIST or LPRINT in NODDY since the text is stored in full pages. You will also have to set your printer to accept a line length of 39 characters. You will find instructions in the manual supplied with your printer on how to change line length. If you have an EPSON type printer, for example, the command you would use is:

```
LPRINT CHR$(27);"Q";CHR$(39)
```

To print out a page of NODDY you simply type *LIST followed by the page title with a full stop. For example:

```
*L TITLES.
```

The computer will print out the page called TITLES.

Noddy is a new and evolving language where there are few rules to govern the way in which programs are written. We have attempted to give you some guidelines as to the way to procede. However they are only guidelines; it is for you to develop your own programming technique. The applications of NODDY are only limited by your imagination.
(The NODDY commands on the MTX are a subset of the complete language as described in the NODDY report (1982).)

PART 3

GRAPHICS

The MTX 500 is capable of very sophisticated graphic effects. You will be able to control the graphics screen in a variety of ways, changing its size shape and colour, as well as designing complex animation programs.

Until now you have been using text screens with characters like a,b,c etc. However, the MTX is capable of high resolution graphics using its graphics screens. There are two distinct types of screen; The text screen which is 40 columns wide and 24 lines deep and the graphics screen which is 32 columns by 24 lines. Text can be written to a graphics screen but graphics cannot be written to a text screen.

It is important, even if you are familiar with graphics production on other machines, to follow this part of the course very carefully. MTX graphics are designed to use a few interactive commands rather than a large number of commands which operate alone. Though this can make your graphics programs simple to set up, it does mean that you have to have a thorough understanding of the commands.

The graphics manual is split into five sections: Controlling Text, Controlling Graphics, BASIC Graphics, Further Graphics and Animation.The first concerns the control of the text screen.

CONTROLLING THE TEXT SCREEN

Though we are introducing these controls as text screen control as you will see they are used interactively with the other graphics commands. We will first of all give you the simple uses of the words and then in the final section draw them together in sample programs where their interactive use is explained.

CLS — The CLS key is used as in many other applications in MTX BASIC to clear the screen to begin a new task. The command CLS, however, can also be used as a command within your program to carry out a similar function.

CSR x,y — The command CSR (cursor) places the cursor on the screen at the coordinates x,y.

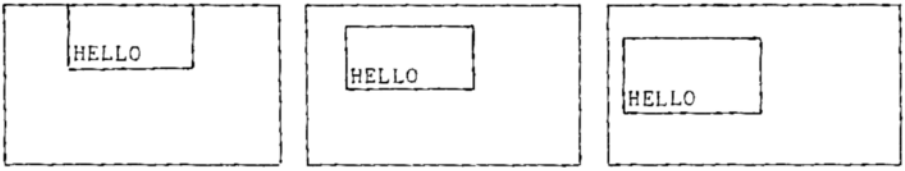
To illustrate the use of these commands type in the following program line and run it.

```
10 CLS:CSR 10,10:PRINT "HELLO"
```

When you run this line the screen is cleared, the cursor moved to position 10,10 (near the middle of the screen) and HELLO printed from this position.

Use the CLS and CSR commands to print text on the screen in different positions. By observing the effect of the coordinates you can develop a mental map of how the screen is divided. This will be a useful skill to develop for the more advanced stages of graphics production. There are further notes in the reference section.

VS (Virtual Screen)
All print commands are relative to the screen you are using. If you have a small TV screen the coordinates 10,10 would be in the same relative position as 10,10 on a large TV screen. The MTX has an inbuilt method to allow you to create smaller screens within your screen. These are called virtual screens. The coordinates 10,10 would also be in the same relative position within your virtual screen. This is demonstrated in the three screens below.



MTX BASIC uses four virtual screens. The editor is VS 0 and consists of four lines which behave as a single line. The list screen is VS 1 and consists of 19 lines, the message screen is VS 7 and consists of one line at the bottom of the screen. The whole screen is called VS 5.

VS 1 LIST SCREEN 19 LINES

VS 5
(whole screen)

VS 0 EDITOR 4 LINES
VS 7 EDITOR 1 LINE

CRVS n,t,x,y,w,h,s
This is the command used to create your own virtual screens. The information is placed into the computer in the form of a parameter statement rather like sound:

CRVS n is the VS identification number in the range 0-7*
t screen type (0 for text and 1 for graphics)
x is a coordinate of the top left hand corner of the VS
y is a coordinate of the top left hand corner of the VS
w width of screen in characters
l depth of screen in lines
s the number of characters which exist in one line of the type of screen in `t' (40 for text screen and 32 for graphic

* NB do not use VS 0,1,5 or 7 since these are used by BASIC itself as identifiers.
VS 4 is used by BASIC for its full graphics screen.
If you should create one of these then it will be redefined whenever you return to the BASIC Ready.

A simple example of a VS would be to define a block 10 characters by 10 characters in the centre of the screen:

```
10 CRVS 2,0,20,10,10,10,40
20 VS 2
30 DSI
```

Line 10 defines the screen, line 20 selects the VS number you wish to use, this should agree with the first parameter in line 10 and line 30 introduces a new command DSI. DSI (Direct Screen Input) tells the computer to direct input from the keyboard to the new VS. If you now type information to the screen you will be able to see where the screen is located and its size.

Now press the <CTRL> key at the same time as the hat key <^> the cursor will appear in the VS. You can now use the cursor keypad to edit the information in your VS. If you switch from one VS to another the cursor will be exactly where you left it so that you can easily carry on inputting from where you left off.

Take this opportunity to try some of the other keyboard controls.

The <PAGE> key is used to decide whether you are in page or scroll mode. In page mode when you reach the bottom of the page the cursor moves to the top of the page for the next page of input. In scroll mode when you reach the bottom of the screen the information you have typed scrolls up. Switch from one to the other to see how the different modes work.

Experiment with <ESC>I and <ESC>J. (Unlike the <CTRL> key in this case type <ESC> followed by I or J). In the reference section there is a list of control characters and escape sequences which you will find useful.

//
EXERCISE 27 VIRTUAL SCREENS

Set up three VS to take your name address and date of birth. You will have to estimate the ammount of space you will need in each case to make sure that they do not overlap.

NAME

ADDRESS

DATE OF BIRTH

//

The two remaining text control commands PAPER and INK were dealt with in the BASIC tutor. You should read through this section if you have not already done so.

Before moving on to the graphic section you should be aware that the default screen in operation when you switch on is a text screen. Before you can begin any graphics you first of all have to define a graphics screen. You will see in the sample programs at the end of this chapter that either a special screen is set up using a CRVS command or VS 4 is selected. This is usually combined with a CLS command. (eg 10 VS 4:CLS)

BASIC GRAPHICS

The commands used in this section are those to be found in standard BASIC graphics. They are used to plot points,lines, arcs, circles and so on.

PLOT x,y is used to plot a pixel(point) at the coordinates x,y.

LINE x,y,p,q draws a line from the coordinates x,y to p,q.

CIRCLE x,y,r draws a circle of radius r with centre x,y.
We have included a small program to show you how these work.

```
10 VS 4
20 CLS
30 FOR I = 1 TO 191 STEP 1
40 PLOT I,I
50 NEXT I
60 CIRCLE 100,100,50
70 INPUT A$
80 IF A$ = "S" THEN STOP ELSE GOTO 10
```

Lines 70 and 80 are important since the computer completes the program in a fraction of a second. The two lines can be replaced by the single line 70 PAUSE 10000. This line will show the effect for ten seconds before returning to BASIC.

Try experimenting with this program. For example you could insert lines:

65 LINE 10,20,150,170
68 LINE 35,150,170,55

Design your own programs which draw circles and lines on the screen. Try to become sufficiently familiar with the commands that you are able to plot on the screen exactly what you intend, without trial and error type guesses.

FURTHER GRAPHICS

In BASIC graphics you have learnt how to plot lines on the screen. In recent years the development of TURTLE graphics has led to an interest in interactive graphics (LOGO for example). Your MTX has the ability to handle this type of program but before we show you the type of program that enables you to set up LOGO-like graphics we will look at the commands which achieve this.

There are four commands which we will be using; ANGLE, PHI, DRAW and ARC. The first three of these we will deal with together. They are used to determine the direction of the lines or patterns to be drawn. The computer remembers a direction which is set by the ANGLE and PHI commands.

ANGLE (radians)

The ANGLE command sets the initial orientation of the computer from a zero value in the horizontal plain through 360 degrees. The values of ANGLE are given to the computer in radians which are converted by the formulae:

To obtain radians:

$$R = \frac{2 \times PI}{360} * D$$

To obtain degrees:

$$D = \frac{360\ R}{2 \times PI}$$

There are several steps which are required to design your programs with accuracy. The first involves working out the initial value of ANGLE. Remember this sets the initial direction given to the computer. The value of 0 would set the initial direction as horizontally across the screen to the right. As you add radians to this the angle with the horizontal is made larger and the intial direction is moved in an anti clockwise direction. You can use ANGLE therefore, to rotate your pattern or shape.

You may not be accustomed to thinking in RADIANS and so we have designed a short program to convert degrees to radians:

```
10 INPUT "TYPE IN THE ANGLE    ";A
20 LET A=A*(2*(PI/360))
30 PRINT "PHI VALUE=    ";A
40 PRINT:PRINT:PRINT:PRINT
50 PRINT "DO YOU WANT ANOTHER NUMBER?"
60 INPUT "Y FOR YES N FOR NO     ";B$
70 IF B$="Y" THEN GOTO 10 ELSE GOTO 80
80 CLS:STOP
```

Amend the program to carry out the reverse calculation from RADIANS to degrees.

The second step is to use the PHI command. Each time PHI is encountered, its angle is added to the direction already held by the computer. We have given you an example of this in the program below.

```
10 VS 4:CLS
20 ANGLE 0
30 FOR I=1 TO 10
40 PHI .1
50 PLOT 120,100
60 DRAW 50
70 PRINT ,,,I
80 NEXT I
90 GOTO 90
```

In this program you will see that as each FOR loop is executed an additional PHI is added, changing the direction of line drawn from the original plot position. This shows you the simplest form of relationship between ANGLE and PHI. The other programs later in this section show you a more dynamic relationship where the two commands combine to draw arcs and spirals.

DRAW x
Draws a line of length x from current plot position in direction set by the other two commands. (ANGLE and PHI).

We have included three programs to help you to see how these commands operate.

```
10 VS 4:CLS
20 ANGLE 0
30 PLOT 100,20
40 FOR I=1 TO 8
50 DRAW 70
60 PHI PI/4
70 PAUSE 1000
80 NEXT I
```

By changing lines 40 and 60 you can make any symmetrical geometric shape using this program. The number of sides is decided in the FOR statement and the size of the angle in 60. PI/4 for example is equivalent to an angle of 45'. Using this method to produce a square the line 40 would be amended to draw 4 sides and PHI would be PI/2.

```
10 VS 4:CLS                    Clear the graphics screen.
20 PLOT 100,100                Set starting position.
30 ANGLE 0                     Set initial direction.
40 FOR I = 0 TO 1 STEP .01
50 DRAW 7                      Draw a line of length 7.
60 PHI I                       Add angle I to direction.
70 NEXT I
```

As the value of I changes PHI is altered in line 60 thus producing a spiral effect. Try changing this program by altering the values for ANGLE and DRAW and see what happens. If we change the size of the step in line 40 to .001 then this smaller step produces a bigger spiral. You will find that to fit the new spiral on the screen you will have to reduce the line length to less than 2.2 in line 50. The alternative to this would be to plot a lower position in line 20.

//

EXERCISE 28 ANGLE PHI AND DRAW.
By changing line 40 in the above program make the spiral reverse by beginning in the centre and spiralling slowly outwards.

//

Make the following alterations to the above program one at a time and at each stage run the program to see the effect.

Move the plot command at line 20 to within the FOR loop at line 45.

Now insert a test at line 65 to prevent the program from stopping.

```
65 IF I=1 THEN GOTO 45
```

The second program uses the same principle to set up a continuous pattern. Again try amending the program to make the effect bigger and smaller and move the starting point around the screen.

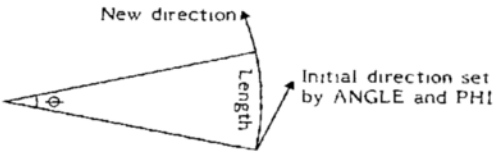
```
10 VS 4:CLS
20 PLOT 200,55
30 LET I=0
40 ANGLE 0
50 DRAW 1
60 LET I=I-.1
70 PHI I
80 GOTO 50
```

The commands you have used so far will give you increased control over the displays you can produce. The curves in the two programs above are useful to produce spirals; (the curve gets steeper and steeper).

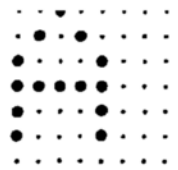
ARC x,theta
Draws an arc length x while turning through an angle theta. In the program below we have used ARC to draw a series of lines from the plotted position in a spiral manner. By adding another loop and reversing the effect try to make the shape into an 11 leaved flower.

DIAGRAM

```
10 VS 4:CLS
20 ANGLE 0
30 FOR I=1 TO 11
40 PLOT 120,100
50 ARC 100,2: PHI 2
60 PAUSE 1000
70 NEXT I
```



CONTROLLING THE GRAPHICS SCREEN
Before moving on to the creation of more complex shapes and sprites it is well to remember how your graphics and text screens work. They consist in both cases of a series of points called pixels which can be switched on and off. When working in the default setting of the text screen, the background (paper) is blue and text (ink) is white. When you press a text key 'A' for example the pixels which make up 'A' are switched from background PAPER to foreground INK and the letter A appears. The letter A is a pattern made up of pixels within an eight rows by eight columns matrix.



Notice that there are spaces below and to the right of the letter to stop adjacent characters merging together.

You may have noticed already that the size of characters on the graphics screen appears larger than on the text screen. This is because in order to place 40 characters on the text screen the computer ignores the two rightmost columns of dots. If you look at a letter 'A' displayed on the screen you may be able to see the dots which make up the pattern. The text screen can therefore be considered as a matrix of dots large enough to display 24 rows of 40 characters. The number of dots can be calculated as:

```
40 X 24 characters
          =   (40x6) X (24x8) dots
          =   240 X 192   dots
```

The ASCII characters are simply an internationally accepted set of patterns including letters numbers and symbols each of which is associated with a unique number called its ASCII code.(see reference section)

Type in and run the program below.

```
10 VS 4
20 CLS
30 PRINT "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
40 DSI
```

This allows you to type characters on the graphics screen and you should be able to see the dot pattern of each character and the gap between them caused by displaying the complete 8X8 pattern.

The above calculation for the graphics screen is:

```
32 X 24 characters
          =   (32x8) X (24x8) dots
          =    256 X 192 dots
```

When using graphics therefore you can think of the screen as being made up of a 256 X 192 matrix, where the dots are selected in commands such as PLOT by considering the screen as a graph with the axes along the bottom and left hand side and the required dot being specified as coordinates.

In the text mode we set the foreground and background colours on the screen using the commands PAPER and INK and the only patterns which we displayed there were the pre-defined ASCII characters. The GRAPHICS screen is much more flexible in terms of colour and patterns and accordingly we need a number of extra commands.

When we used PLOT, DRAW and ARC, we were actually changing the colour of individual pixels. When you clear the screen with the CLS command you are setting all of the pixels to the colour chosen by the PAPER command. Typing or PLOTting on the screen changes some of the pixels to the colour chosen by the INK command thereby creating a pattern. Changing colour in this way gives the illusion of switching pixels on and off.

Aspects of control such as defining screens and their manipulation are the same for text and graphics. However, the use of colour in graphics is much more complex and sophisticated. The commands COLOUR and ATTR are used to set up parameters. They enable you to have greater control over the colours you produce.

COLOUR p,n (Graphics only)
Colour is the command which determines which colour is used.

p is the parameter
n is the colour

The parameter concerns which areas of the screen are to be of the colour defined by 'n'. The values of n are as in the commands PAPER and INK. The values of p are explained below. To understand graphics colour you have to be aware of the composition of the graphics screen which we have just explained to you. You may for example be writing text to the screen in which case you would use the normal paper and ink values.

p = 0 = print paper
p = 1 = print ink

The pixels which are changed from the PAPER to INK colour when characters are sent to the screen are determined by the ASCII codes.

When using graphics commands to plot or manipulate the graphics screen, however, each pixel is potentially treated individually. There are 256 by 192 pixels on your graphics screen. Each of these therefore, can have the same colour properties as the text screen. To control these you would use the non print paper and ink commands. In this case pixels which you have plotted would take on the colour defined by the parameters 2 and 3.

p = 2 = non-print(plot) paper
p = 3 = non-print(plot) ink

The final parameter is concerned with the remainder of the screen. The use of this value will make the border around the graphics screen the colour 'n'.

p = 4 = border colour

Try this program and vary the colours in the lines 20,30 and 40 using the chart in CHAPTER 1.

```
10 VS 4:CLS
20 COLOUR 2,5
30 COLOUR 3,3
40 COLOUR 4,6
50 ANGLE 0
70 PLOT 120,100
80 DRAW 50
90 PHI .2
100 GOTO 70
```

ATTR p,state (graphic only)
The second graphics command involves the further manipulation of the pixels as set by the COLOUR command. The command ATTR can help you achieve very sophisticated graphic effects by changing the properties of the pixels you have activated by typing or plotting information on the screen. We have written a short program to show you how the command works:

```
10 VS 4:CLS
20 INPUT "ATTR P N ?    ";P,N
30 ATTR P,N
40 DSI
50 GOTO 20
```

If you run this program it will ask you for values of p and n. 'n' simply switches the ATTR on and off where 1 is on and 0 is off. Try typing over characters to see what happens for different attributes. Continue the program loop by <RET> to exit from the DSI command and select another ATTRibute.

p = 0 ; inverse print ATTR

If you set the n value to 1 you will see that the characters you print are reversed so that the characters are printed in the paper colour and the paper in the ink colour. If you return to the home position then type another character the point of overlap in the characters is reversed. This can give you interesting pattern effects.

If an attribute is switched on, it can be switched off by typing in p,0. (p=1,2,3 or 4).
The ATTR settings are switched on and off rather like using the PAGE key to switch from mode to mode.

Switch off attribute 0.

Enter 0,0 as the p,n values.

p = 1 ; overprint ATTR

If you now switch this ATTR on by typing in 1,1 and type in 10 'D's and ten spaces followed by 10 'D's and so on. Return to the home position and hold down the D key. 'D's will be replaced with spaces, and spaces with 'D's. With this attribute switched on, points plotted on top of other points will always have the effect of unplotting the point. This is why a D typed on top of another D erases it.

The ATTR commands are not used exclusively and can be merged to combine effects. If you now return to the program and input 0,1 this will have the effect of leaving 1,1 switched on but adding the inverting paper and ink effect in 0,1.

Now try switching on and off the following ATTR effects and experiment merging the different commands.

p = 2 ; unplot ATTR

When this is set points will be unplotted rather than plotted. In other words, the points will be plotted in the paper colour rather than the ink colour.

p = 3 ; over plot ATTR

If this is set:

A) During plotting, plots a point if it wasn't already there and leaves points already plotted unchanged. This allows characters to be written over each other.

B) During CLS and other functions, the text is unchanged but colours can change. This is useful for changing paper and ink while leaving text intact.

If both inverse plot and over plot attritubes are set then the effect during plotting is to do nothing on the screen. This can be used to move plot position but leave the screen the same. You can use this to guide the PLOT SPRITE (see below) around the screen.

ANIMATION

Animation can be achieved on the MTX by the use of SPRITES.

A sprite can be thought of as a small drawing board on which objects can be drawn. By moving the drawing board, the object drawn on it will appear to move around in front of the graphics screen.
Type in and run the following program which should produce an arrow moving from left to right across the screen. Don't worry about how the program works at this stage.

```
10 VS 4:CLS
20 CTLSPR 0,1
30 CTLSPR 2,1
40 CTLSPR 3,1
50 CTLSPR 5,1
60 CTLSPR 6,1
70 GENPAT 3,1,24,4,2,255,255,2,4,24
80 SPRITE 1,1,1,100,10,0,1
90 GOTO 90
```

The sprites are similar to characters and are either 8x8 pixels or 16x16 pixels but unlike characters a sprite can have only one colour.

There are 32 sprites which are numbered from 0 to 31 and are arranged as in the diagram below:

Each of the pictures in the diagram represents a single display plane within which a sprite can move. The sprites are arranged in this way so that you can build up animations which have depth with sprites able to pass in front of and behind each other.

Since each sprite can be a different colour, multi-coloured objects can be created by overlaying several sprites. Beware however that a maximum of 4 sprites are allowed in any horizontal row before the results become unpredictable.

If you study the diagram above you will see that the PAPER and INK plane is behind the sprites. As the sprites move, the background remains static. In order that your graphic displays can appear real ie. where the sprites enter and leave the screen, the sprite planes are bigger than the screen. This means that the sprites you are to use need not suddenly appear but can be waiting in the wings offscreen until the program calls them into action. Circling sprites can also be defined which apparently orbit your television so that when they disappear off one side they will reappear on the other some time later.

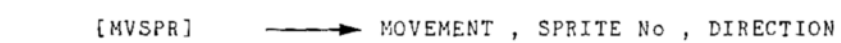
Change line 80 in the above program to

```
80 SPRITE 1,1,1,100,120,0,1
```

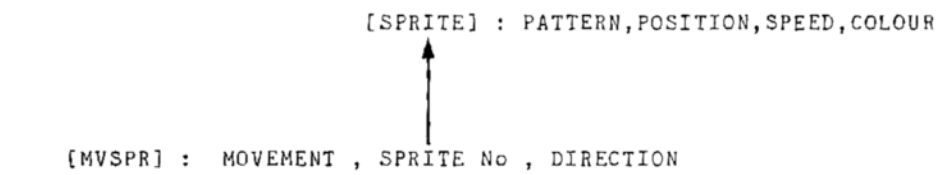
When you run this program, wait a few seconds and you will see the sprite orbiting.
The commands used to set up these complex pictures are interactive. That is to say each affects the other to control the activity on the screen. You have already seen how the pixels on the text and graphics screens are manipulated to give interesting patterns and effects. These are used to form the background to your animation. The sprite commands are used in much the same way to define each sprite plane in turn to build up the total picture.

Before giving you the details of the commands, we are going to build up a diagram to show you how they relate together:

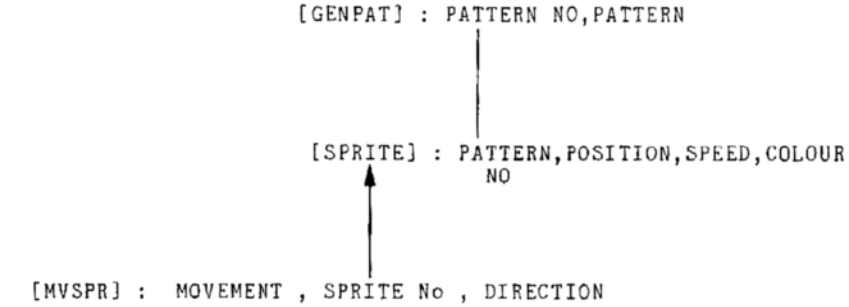
The whole basis of animation is that your shapes (sprites) can move. The first command we will look at therefore, is MVSPR (movesprite). In each command we have to tell the computer which sprite we are referring to (numbers 0-31) and in the MVSPR command we can tell the computer how to move the sprite and in which direction.



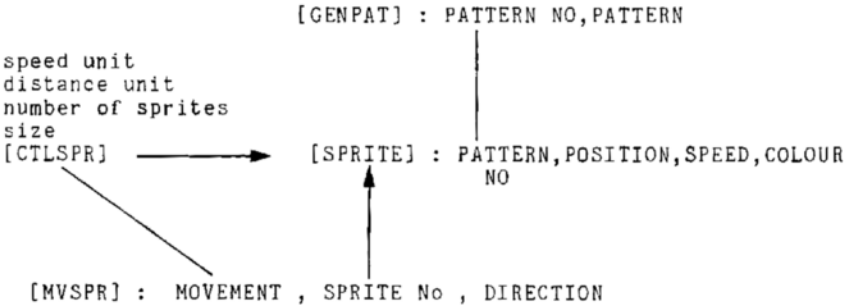
Though this command has told us which sprite plane is to be used the sprite has no shape or colour and so at this stage you could not see it. We therefore use the SPRITE command to define the sprite.



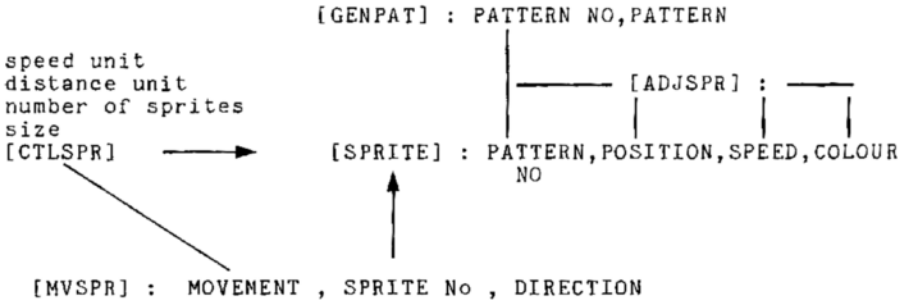
In the SPRITE command the sprite is given a position, a speed, a colour and a pattern number. The pattern number selects a shape for the sprite which has been defined using the GENPAT command.



Having set up the sprite you now have to control it. Each time the MVSPR command is used, it tells the sprite to move one step in the given direction. The step size however is specified in the CTLSPR command, as are other parameters such as the size of the sprites, the unit of speed and how many sprites we actually want to use.



Notice that the CTLSPR command affects all of the sprites,and that the SPRITE command only affects an individual sprite. If we wish to change the speed colour or position of an individual sprite we don't repeat the SPRITE command but instead make adjustments to it using the ADJSPR command. This has the affect of altering a single parameter by specifying which sprite, which parameter and its new value.



Remember there are two sizes of sprite. The first is 8 columns by 8 rows of pixels whilst the larger version is 16 by 16. The first of these requires only one GENPAT statement whilst the larger requires 4, one to define each 8 by 8 quadrant of the whole shape. All sprites in use at any time must be the same size which is selected using the CTLSPR command. Having defined the size in this way you can make it grow to twice its size by using the MAGNIFY parameter again in the CTLSPR command.

In CTLSPR we defined the distance unit. This is the number of pixels to be moved during a MVSPR command. CLTSPR can also allow a number of sprites to move by themselves. These sprites are set up by GENPAT and SPRITE as before but they now need to be given a speed. The speed of a sprite moving in this way is determined by the speed unit set up in CTLSPR. The CTLSPR speed unit sets the step size (ie pixels per second). The actual speed is then determined by the SPRITE command which sets the number of steps. Therefore, if the step is set at 20 pixels per second in [CTLSPR] and 5 units in [SPRITE] the final speed would be:

20 pixels x 5 units = 100 pixels per sec

We are now going to build up a program step by step to create a sprite and make it move. You will find it useful to refer to the introduction and diagrams above to make sure that you understand each step. As we use each command we will give you all of the parameters which can be selected.

GRAPHICS SAMPLE PROGRAM
Remember the first step when writing graphics programs is to set up a graphics screen:

```
10 VS 4:CLS
20 CTLSPR 2,1
30 GENPAT 3,0,255,129,129,129,129,129,129,255
40 SPRITE 1,0,128,96,0,0,1
50 CTLSPR 1,1
60 LET Y=ASC(INKEY$)-48
70 IF Y>8 OR Y<1 THEN GOTO 60
80 MVSPR 9,1,Y
90 GOTO 60
```

Line 20 CTLSPR is used to tell the computer that there is going to be only one sprite in the program. If you look at COMMAND 1 below you will see that the CTLSPR command works like this:

The value for x varies as you can see for each parameter. In the case of line 20 parameter 2 tells the computer how many sprites to expect and the x value of 1 indicates that only one is to be used. This parameter is used like the DIM statement in that it is informing the computer of the amount of space required.

Line 30 defines the pattern for the sprite. Read through the details in COMMAND 2. You will see that GENPAT 3 defines the pattern for an 8 by 8 sprite.

```
30 GENPAT 3,0
```

The 0 is the pattern number. Any sprite which is assigned the pattern number zero will be given this pattern.

```
30 GENPAT 3,0,255,129,129,129,129,129,129,255
              r1  r2  r3  r4  r5  r6  r7  r8
```

The rest of the numbers ,r1 to r8 above, define the pattern. Each row of the sprite is defined by one of the numbers i.e. r1 defines the top row,r2 the second and so on.

To design a sprite, first draw the pattern in an 8x8 matrix on graph paper.

```
128 64 32 16  8  4  2  1

  0  0  0  1  1  0  0  0      r1=24
  0  0  0  0  0  1  0  0      r2=4
  0  0  0  0  0  0  1  0      r3=2
  1  1  1  1  1  1  1  1      r4=255
  1  1  1  1  1  1  1  1      r5=255
  0  0  0  0  0  0  1  0      r6=2
  0  0  0  0  0  1  0  0      r7=4
  0  0  0  1  1  0  0  0      r8=24
```

This pattern is the orbiting arrow in the example above. To find the numbers in the GENPAT statement, just add up the numbers at the top of any column which has a 1 in it.

To give you experience setting up GENPAT statements input the program below. You will be able to input this program with the other still in working memory since it starts at line 100. When you run it the cursor will go to the top of a virtual screen and the number 4 will appear with a question mark. The 4 indicates that you are inputting the GENPAT statement 4 and the question mark is asking you to input 8 numbers between 0 and 255. Each number has to be separated by a comma.

Try the following line first:

```
255,129,129,129,129,129,129,255
```

When you <RET> the sprite will appear at the foot of the screen and the top left hand corner will take on a square shape. On the virtual screen HAPPY ? will appear. This is to give you a chance to change the line if you wish. If you are happy press "Y" and go on to GENPAT 5 if not press "N" and do 4 again. Now experiment with different numbers for the other lines. When you have completed lines 4,5,6 and 7 you have completed a sprite.

```
100 CRVS 4,1,1,3,30,10,0
110 VS 4
120 CLS
130 CTLSPR 2,32
140 CTLSPR 5,0
150 CTLSPR 6,3
160 FOR I=4 TO 7 STEP 1
170 PRINT I
180 INPUT A,B,C,D,E,F,G,H
190 GENPAT I,0,A,B,C,D,E,F,G,H
200 SPRITE 1,0,100,30,0,0,1
210 INPUT "HAPPY  ?";A$
220 IF A$="Y" THEN GOTO 230 ELSE GOTO 170
230 NEXT I
240 GOTO 160
```

To return to the explanation of the program, line 40 is the command which sets the parameters to control the sprite.

```
40 SPRITE 1,0,128,96,0,0,1
```

The first digit is the sprite number which tells the computer which sprite plane this sprite will operate in. The second is the pattern number which was set in the GENPAT statement. The number 128 is the position of the centre of the sprite on the x axis and 96 sets the coordinate on the Y axis.(The coordinates are set as in PLOT with 0,0 being the bottom left hand corner of the screen.)

The fifth and sixth digits set the speed of the sprite, the first being the speed along the x axis the second the speed along the Y axis. In this case we do not want the sprite to travel independently and so no speed instruction is given. The final number in the statement sets the colour at 1.

Line 50 of our program is used to give the sprite its instructions about the way it is going to move. If you look in COMMAND 1 the instruction 1,1 means that the sprite will move 1 pixel at a time when requested by a key depression. (NB the Auto Repeat function affects the plotting in that the sprite will move at the speed of auto repeat if it held down.)

```
50 CTLSPR 1,1
```

Lines 60 and 70 are used to allocate keys to move the sprite in different directions. Each direction is allocated a separate key in 60 (the -48 is to reduce the ASCII code to a range of 1-8) and if any other key is pressed a loop places control back in line 60.

```
60 LET Y=ASC(INKEY$)-48
70 IF Y>8 OR Y<1 THEN GOTO 60
```

Line 80 uses COMMAND 4 to instruct the computer to move in response to the eight key depressions. The CTLSPR command was used to set movement on request. The MVSPR command now instructs the computer to move the sprite in one of eight directions. The MVSPR uses a bit pattern rather like the GENPAT statement to instruct the computer about the nature of the movement. This enables you to give a series of instruction in one digit by adding the options together.(See COMMAND 4)

```
80 MVSPR 9,1,Y
```

The final line 90 takes control back to line 60 to wait for the next input.

```
90 GOTO 60
```

There are two other commands which we haven't used. These are ADJSPR and VIEW.

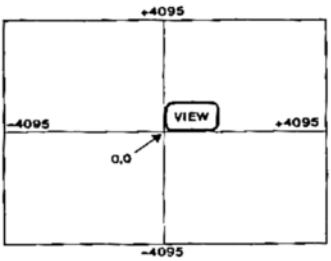
ADJSPR is used to alter any one of the values which have previously been set up by the SPRITE command. For example if we wish to change the colour of sprite number 3 from 1 to 5, we should use the command

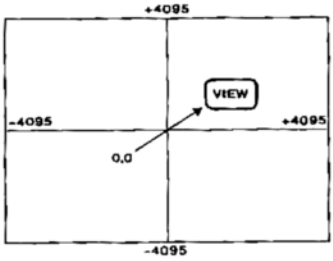
```
ADJSPR 1,3,5
```

This command has advantages over re-using the SPRITE command because it is faster in that only one parameter is changed at a time, and also we don't have to worry about altering any of the other parameters.

The VIEW command has the effect of looking through a window in front of the sprite planes (8192X8192 pixels) where the window is your television screen (256X192 pixels).

Initially the window is located near the centre of the sprite planes with location 0,0 in the graphics screen equal to location 0,0 in the sprite plane.





Whereas the MVSPR command moves an individual sprite relative to the graphics screen, the VIEW command moves the graphics screen relative to all of the sprites. This means that complicated sprite patterns made up of many different sprites can easily be moved. Also sprites can be hidden in the sprite planes in fixed locations such that they will only come into view if the window is moved over them.

We have tried to give you an overview of the way in which MTX series graphics works. You will need to experiment yourself to become an expert. The COMMAND words below will give you all that you need to know, but the descriptions cannot tell you how they interact. This you will have to find out for yourself. We have listed two more programs for you to input. Try to understand how they work and then try to change and add to them to produce different effects.

EXAMPLE PROGRAM

```
10 CTLSPR 0,6
20 CTLSPR 2,10
30 CTLSPR 6,3
40 GENPAT 4,0,1,0,1,2,3,15,1,3:GENPAT 5,0,2,3,2,6,6,0,0,0:
GENPAT 6,0,64,128,192,160,224,248,192,224:GENPAT 7,0,32,22,
32,48,48,0,0,0
50 SPRITE 1,0,0,0,0,0,6
60 CTLSPR 4,1
70 CRVS 6,1,0,0,32,24,0:PAPER 15:COLOUR 4,6:INK 1:CLS
80 ATTR 3,0:ATTR 2,0
90 PLOT 80,80:ANGLE 0
100 FOR I=1 TO 11
110 ARC 100,2:PHI 2
120 NEXT I
```

Line 10 controls the speed of the sprite.If you alter this refering to COMMAND 1 you will be able to speed up and slow down the sprite.

You can make the sprite grow by changing line 30 to:

```
30 CTLSPR 6,3
```

You can also extend the program by amending line 80 to:

```
165 ATTR 3,1:ATTR 2,1
```

and then add:

```
200 ATTR 2,0
210 CTLSPR 5,3
220 CTLSPR 0,1
230 LET S=25
240 SPRITE 3,0,100,130,S,0,2
250 FOR W=1 TO 20
260 LET Y=0
270 FOR Z=1 TO 8
280 LET Y=Y+1
290 FOR X=1 TO 25
300 NEXT
310 LET D=Y-8
320 MVSPR 12,3,D
330 ADJSPR 1,3,D+2
340 NEXT
350 LET S=S+5
360 ADJSPR 4,3,S
370 NEXT
```

EXAMPLE PROGRAM

```
10 VS 4:CLS
20 PAPER 1:INK 7:CLS:ATTR 3,1
30 FOR X=0 TO 255
40 LINE X,191,255-X,0
50 NEXT
60 FOR Y=1 TO 190
70 LINE 0,Y,255,191-Y
80 NEXT
90 FOR K=2 TO 94 STEP 4
100 CIRCLE 128,96,K
110 NEXT
120 GOTO 30
```

(NB This program will continue running until you press the BRK key)

COMMAND 1: CTLSPR p,x

p = parameter and can be any of the six below:

0 Speed
1 to 255 to 0 (1 is fastest)

1 Distance
Tells the computer to move the sprite by `x' pixels when requested.

2 Number of sprites
0 to 32 (The number of sprites must be at least 1)

3 Number of circling sprites
Sprites that will orbit when they go off the edge of the screen (must not exceed total number of sprites)

4 Plot sprite
A PLOT SPRITE can be chosen which will subsequently appear whenever a point is plotted. This sprite will move around the screen following any points or lines drawn by the BASIC GRAPHICS commands. This sprite can be any of the 32 defined in the normal way.

5 Number of moving sprites 0 to 32
This is the number of sprites that will move by themselves according to the x-speed and y-speed set in the SPRITE and ADJSPR commands.

6 Magnitude and size

x=0 size 8X8 mag 1
x=1 size 8X8 mag 2
x=2 size 16X16 mag 1
x=3 size 16X16 mag 2

COMMAND 2: GENPAT p,n,d1,d2,d3,d4,d5,d6,d7,d8

The GENPAT command is the command used to generate all types of patterns required by BASIC for characters and SPRITES. There are 5 modes.

1 To redefine an ASCII character. (CODES 32 TO 127)
2 To define a non ASCII character.(CODES 129 TO 154)
3 To define colour for each line of a character.
This only applies to user definable characters with codes 147 to 154.
4 To define an 8 by 8 sprite pattern.
5 To define each quadrant of a 16 by 16 sprite.

User definable characters have codes from 129 to 154.

Mode 1 allows the user to redefine one of the standard ASCII character patterns. Note that the ASCII characters are the ones which are most often used by the computer

Mode 2 allows the user to define his own character patterns without destroying any of the standard ASCII characters.

Mode 3 allows some of these user defineable characters to be further defined by specifying an ink and paper colour for each of the eight rows of the character.

The values for ink and paper are as specified in the table in CHAPTER 1 but in this instance we are specifying two colours (ink and paper) at the same time. Each of d1 to d8 specify a paper and ink colour as a single number:

```
bit  0 1 2 3: 4 5 6 7
     ink    : paper

value = 16 * paper + ink
```

e.g. Red ink on blue paper

```
= RED : BLUE

=BLUE * 16 + RED

=  4  * 16 + 9

=  73
```

MODE	P	N	
1	0	ASCII code (32 to 127)	
2	1	user definable (code 129 to 154)	
3	2		
4	3	pattern number	8 by 8 sprite pattern
5	4	pattern number	16 by 16 NW quarter
	5	pattern number	16 by 16 SW quarter
	6	pattern number	16 by 16 NE quarter
	7	pattern number	16 by 16 SE quarter

COMMAND 3: SPRITE

SPRITE (= create sprite)

SPRITE n,pat,xp,yp,xs,ys,col

n is sprite number 1 to 32
pat is pattern number 0 to 127 (size 0)
0 to 31 (size 1)
xp is position x off centre

yp is position y off centre (in range -4095 to 4095)
0,0 is defined as bottom left hand corner of screen i.e same as for plot.
NB Sprite coordinates are absolute and do not look at virtual screen origins (i.e assume a 32 by 24 graphic screen)

xs is the speed in the x direction range -128 to 127 where 1 unit of speed moves the sprite 1/8 pixel every master speed cycle as set by CTLSPR 0

ys is the speed in the y direction (plus = upwards) range -128 to 127

col = colour 0 to 15

COMMAND 4: MVSPR p,n,d

MVSPR is a general purpose command which combines 4 distinct functions:

p	meaning
1	MOVEMENT
2	PATTERN SELECTION
4	REDIRECT
8	PLOT AT CENTRE

The functions are combined to allow complicated movements to occur whilst using only a single instruction. The type of activity is selected by p as in the table above. If combinations of activities are required, just add the p values together. Some examples are given below.

		eg 1	eg 2	eg 3
MOVE	1	YES	YES	YES
PATTERN	2	NO	YES	NO
REDIRECT	4	NO	YES	YES
PLOT AT CENTRE	8	YES	NO	YES
TOTAL p value		9	7	13

As before n selects the sprite number.

d is slightly more complicated as it must be able to reflect a value for several activities. If d is not in the range of any one of the chosen activities an error will occur.

MOVE (p=1) moves the sprite 1 step in the direction specified by d. The step size is set in CTLSPR 1 and the direction must be in the range 0 to 8 where directions 0 and 8 are the same.

PATTERN changes the sprite pattern to pattern number d. This pattern should have been defined in a GENPAT statement.

REDIRECT picks up the current velocity vector and switches it to the new direction.

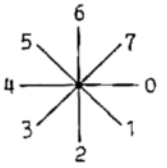
PLOT AT CENTRE causes a point to be plotted at the centre of the sprite specified by n.This is not directly affected by the value of d at all.

COMMAND 5: ADJSPR p,n,v

p	meaning	range of v
0	pattern	0 to 31 (size 1) 0 to 127 size 0
1	colour	0 to 15
2	x pos	0 to 255
3	y pos	0 to 255
4	x speed	0 to 255 (128 to 255 = neg)
5	y speed	0 to 255 (" " ")

COMMAND 6: VIEW direction, distance

directon = 0 to 7
distance = 1 to 255 to 0



GRAPHICS FUNCTIONS

SPK$ (screen peek)
Gives the character at the cursor location on the current text screen.

e.g LET A$ = SPK$

Uses: storing screens

GR$ (x,y,b)
x and y are locations on the virtual screen

b is number of bits read.(If b = 1 equivalent to "POINT" function)

bits are vertical bits ie GR$ (20,190,4) gives a character made up as follows:

```
bit 7       0
bit 6       0
bit 5       0
bit 4       0
Bit 3       pixel at 20,190
bit 2       pixel at 20,189
bit 1       pixel at 20,188
bit 0       pixel at 20,187
```

DSI

direct screen input

Allows you to roam about freely within screen only ending when carriage return is pressed
CTL W = Tab back
CTL] = PMODE
CTL \ = SMODE
CTL ^ = CURSOR ON
CTL _ = CURSOR OFF
CTL D , letter A to O = paper A to O (1 to 15)
CTL F " " ink A to O 1 to 15
ESC I = insert line
ESC J = delete line
ESC K = duplicate line

PART 4
SOUND

Your MTX can produce a wide variety of sounds which can make your programs more interesting and is sufficiently complex that you can use the computer as synthesiser.

Sound is obtained by inputting a sound statement which can be in two forms:

1. DIRECT
   This mode plays a single note until stopped.

2. CONTINUOUS
   Sequences of notes can be played by loading them into a sound buffer.

In each case a sound statement is used to tell the computer what sound you want.

Direct Sound is produced by the statement:

SOUND CHANNEL,FREQUENCY,VOLUME

CHANNEL - There are four channels available. 0,1 and 2 are all pure tone and channel 3 which is a pink noise channel. The use of noise is covered later in this section.

FREQUENCY - Frequency is determined by values in the range 0 to 1023. The sound tables in the appendix gives you the relationship between this value and the frequency produced. The notes produced are also included.

VOLUME - The volume is determined by entering a value between 0 and 15 where 15 is the loudest and 0 is off.

Now try the following sounds:

SOUND 0,200,10 <RET>

Press the two reset buttons and try:

SOUND 1,600,10

and

SOUND 2,900,10

Try the sounds together by entering them one at a time without resetting the computer. You will produce a chord in this way.

Experiment with your own sounds varying the channels, frequencies and volumes. It is useful to refer to the sound tables to monitor the effects of the changes you make and to enable you to understand how the sound chips work.

CONTINUOUS SOUND is produced by a longer statement with seven parameters to enable you to make the sound change in pitch, volume and duration. To produce the continuous sounds the computer loads the statement into a sound buffer.

The sound buffer is a block of memory allocated for use by the continuous sound command. The size of the buffer is chosen by the SBUF command. SBUF 3 for example allocates three blocks for each channel. The default value is two blocks per channel and therefore, if you do not specify a value high enough to accommodate your sound then part or all of your statement will not operate. In this way the SBUF command is similar to the DIM statement. Each block takes 12 bytes per channel so the larger the number of blocks, the less room there is for programming.

Each time a continuous sound command is used, an entry is made in the sound buffer. Each entry is completed before continuing with the next such that a complex sound lasting several minutes can be constructed and left playing whilst other parts of the program are running.

In the example we are going to try we are using only one statement which does not require more than the default setting of two sound buffer blocks. However, to remind you to use the command SBUF we shall first set the buffer to accept 10. You can use any value up to 255.

Type SBUF 10 <RET>

The sound statement looks like this:

SOUND CHAN,FREQ,VOL,FREQ GRADIENT,VOL GRADIENT,TIME,ACTION

We will first go through the statement step by step setting up an example and then give you a series of sample sounds to try. You should then be sufficiently confident to experiment and try your own sounds.

To obtain continuous sound we use an extended version of the sound command. There is an important difference between the values for the different types of input. If you look at the sound tables you will see that there is a column for DIRECT SOUND and a column for SOUND BUFFER. The values for the sound buffer have a greater range to allow you a greater degree of discrimination.

The first information the computer needs is the CHANnel which is in the range 0-2. Pink Noise is generated in channel 3 and since in this case we wish to use a pure sound channel the statement begins:

```
SOUND (0-2),  ie SOUND 1,
```

We then set the FREQuency which is in the range 0-1023 when a sound channel is in use. Frequency determines the pitch of the note and if you examine the sound table you will see that the lower the value you place here the higher the note. We are going to start with a very high note and therefore, we enter a 0 here.

```
SOUND 1,0,
```

The volume of sound when the sound buffer is in use is 0 to 240 and we are going to select a medium volume of 100.

```
SOUND 1,0,100  (if we stop here and press <RET) a tone will
                be heard)
```

If we wish to change the note we give the computer a FREQuency GRADient instruction. This is in the range (-32767 to +32767) The minus values make the note rise the plus values make the note fall. If we placed a 0 here the note will stay the same. However we are going to make the high note fall in pitch and so we enter a value of 10.

```
SOUND 1,0,100,10
```

Remember that increasing the frequency parameter decreases the pitch.)

We can make the volume level change in a similar way by using the VOLume GRADient. The range is (-32767 to +32767) and as before a zero value would give us a continuous level. If we wished to make the volume fade away we would use a minus value. A positive value increases the volume. We are going to use a 0 value so that volume will remain unchanged.

```
SOUND 1,0,100,10,0
```

The computer has to be told how long to sustain the note with the TIME parameter. This is in the range 0-65535 where each unit is 1/64th of a second. We will use a value of 160 which gives us about 2.5 seconds of sound.

```
SOUND 1,0,100,10,0,160,
```

So far we have instructed the computer about the nature of the sound. Where the sounds are to be chained the computer has to be informed about the way the sounds link together. This is achieved using the ACTION parameter. If the sound is not to be linked to the one which follow, a value of 1 is entered. This tells the computer to enter the values in the statement each time the sound is used. However, you may wish to join two sounds together so that they run continuously. In this case a value of 0 is used. This tells the computer to make the starting values of the new sound command equal to the ending values of the previous one. We will first try the sound with a 1 value to stand alone.

```
SOUND 1,0,100,10,0,160,1
```

To edit sound commands more easily it is best to place them in a BASIC program.

```
10 SBUF 10
20 SOUND 1,0,100,10,0,160,1
30 EDIT 20
```

Type run and the sound should be heard. Make sure that you have the volume turned up on your television.

To hear the effect of the action command edit line 20 to replace the action command 1 with 0.

```
20 SOUND 1,0,100,10,0,160,0
```

Run this program a number of times and you will see that the sound continues to descend in pitch each time from the final note in the previous run. Now try again with 1 as the action value and note the difference.

Try the following sounds:

```
1 SOUND 1,5,15,-6,-1,1000,1
2 SOUND 1,5,15,6,-1,1000,1
3 SOUND 1,5,15,0,-12,4000,1
```

CHAINING SOUNDS

Sounds can be linked together in BASIC programs to produce either combinations of sound or sequences. Try the program below and then try to combine some sounds of your own.

```
10 SBUF 10
20 SOUND 1,5,15,1,1,750,1
30 SOUND 2,1,0,10,0,750,1
40 SOUND 3,7,15
```

Type RUN <RET>

Lines 30 and 40 in the program above are a special case since 30 will not run without 40. The VOLume parameter in the statement is set at zero which means sound off. The effect produced is a sudden burst of sound as the VOLume of 15 in line 40 activates the registers in line 30. Sound production is very subtle and therefore, you should experiment as widely as possible using the tables in the appendix. Noise when used in combination with the sound channels can give you very interesting effects like the example above.

There are many ways of incorporating sound production in your programs. In a game for example the graphic effects can be enhanced by adding sounds to them. The best way to do this is to structure your programs so that a series of subroutines are set up to take the program to the appropriate sound when needed. We have listed a useful subroutine here which can be run with your own program:

```
4000 REM SOUND SUBROUTINE
4010 SBUF 2
4020 SOUND 0,100*8,15*64,1,-1,8*64,1
4030 SOUND 1,101*8,15*64,1,-1,8*64,1
4040 LET CHAN=1
4050 GOSUB 5000
4060 SOUND 0,0,0
4070 SOUND 1,0,0
4080 LET DELAY = 400
4090 FOR N=15 TO 0 STEP -1
4100 SOUND 3,4,N
4110 FOR J=0 TO DELAY
4120 NEXT J
4130 LET DELAY =DELAY -30
4140 NEXT N
4150 STOP

5000 REM TEST SUBROUTINE
5010 IF PEEK(CHAN*10+64082) <> PEEK(CHAN*10+64082+4) THEN  GOTO 5010
5020 RETURN
```

The first part of the subroutine is concerned with sound production. The second subroutine at line 5000 is a useful way of controlling your sound production. Line 5010 tests whether the sound in lines 4010, 4020 and 4030 have been completed. When the test is complete control is returned to the main program and in lines 4060 and 4070 the first sound is switched off. The program continues in line 4100 where the noise channel is set up and operated.

As you have seen the production of sound is both subtle and complex. You will need to play with the sounds that we have given to you as well as following the rules outlined in the reference section and this chapter. If you discover a new sound why not share it with the rest of us.

PART 5

ASSEMBLER

This section does not attempt to teach you how to use ASSEMBLY LANGUAGE (machine code) but rather how to interface assembly language to MTX BASIC using the MTX assembler.

The assembler is invoked by telling the computer that you want to write some machine code and where you want to put it.

Look at the program below. Lines 10,20 ,40, and 50 are normal BASIC lines. Line 30, however has been created by the assembler. If you type in lines 10,20,40 and 50 and then follow the instructions you will see how this is done.

```
10 PRINT "START OF PROG"
20 POKE 40000,5

30 CODE

8029        LD A,(40000)
802C        INC A
802D        LD (40000),A
8030        RET

Symbols:

40 PRINT PEEK(40000)
50 PRINT "END"
```

Type ASSEM 30 <RET>

Assemble will appear at the bottom of the screen.

Type <RET> again.

The screen should look like this:

```
8029            RET
```

Insert

INSERT tells you that you are in the insert mode. ie If you type in assembly language,it will be inserted without destroying anything already there.

8029 is the address at which the code will be inserted. The number is a hexadecimal number.

RET is the instruction which currently occupies the address.

Now type LD A,(40000) <RET>

802C RET

Insert

will appear.

This tells you that the next line will be inserted at the address 802C and that the location is currently occupied by a RET instruction.

Type INC A

The screen will now appear:

802C INC A RET

Press the <RET> key and the message Bad Code will appear. Obviously you need to remove the RET from the end of the line. To do this use the DEL key on the cursor keypad.

Now press <RET> again and type in

LD(40000),A <RET>

Press <CLS> followed by <RET> and Assemble <RET> will reappear as you are returned to the assembler. You can clear the screen and <RET> to the assembler in either Insert or Edit mode.

To list your program you first have to move the program pointer to the Top of your program.

This is done by typing T <RET>.

Now type L <RET>.

The program pointer remembers the last position and you program is listed to that point.

```
8029    LD A,(40000)
802C    INC A
802D    LD(40000),A
8030    RET
```

Symbols:

To return to BASIC you clear the screen and press return:

<CLS> <RET>

Ready will reappear.

You are now ready to list your entire program and it should appear as the program listed earlier with code line 30 inserted.

If you run the program the screen should appear like this

```
START OF PROG
 6
END
```

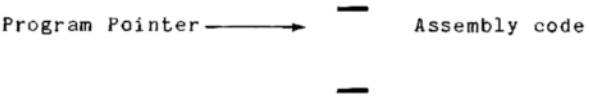
SUMMARY

The assembler is invoked by typing ASSEM <Line Number>

To return to basic type <CLS> followed by <RET>

To insert code enter the assembler and press <RET> and to stop inserting <CLS> followed by <RET>.

To list your code whilst in the assembler type T <RET> followed by L <RET>.



The program pointer remembers the line you are editing or the point where you are inserting text into the program.

T moves the pointer to the top of the program. You would probably want to do this before listing so that you need not remember the address at which the program starts.

Insert Mode

In the insert mode, any lines typed into the computer will be inserted at the address on the left of the screen. The correct ammount of space in memory will be made for each line as it is entered.

There are four ways of entering the insert mode:

1 <RET> enters at the program pointer position.
2 £n enters at the HEX address `n'
3 n enters at the decimal address `n'
4 Label enters at the label if it exists.

To exit from insert mode <CLS> <RET>

Edit Mode

In the edit mode each line entered replaces the line originally displayed. In this way it differs from the insert mode where lines are inserted without altering what is already there.

As with the insert mode there are four ways of entering.

1 E<RET> enters the editor at the program pointer
2 E £n enters the editor at the Hex address `n'
3 E n enters the editor at the decimal address `n'
4 E label enters the editor at the label specified if it exists.

NB If a label E exists then if E <RET> is typed the insert mode is entered at label E rather than the editor at the program pointer.

List

1 L<RET> lists the program from the program pointer
2 L £n lists the program from the HEX address `n'
3 n lists the program from the decimal address `n'
4 L label lists the program from the label if it exists.

NB as with the edit mode a label L will lead to L<RET> entering the insert mode at L instead of listing from the pointer.

Delete

Lines can be deleted either in the edit or insert modes. When a line is displayed the cursor appears between the address and the code. If you type EOL,or type spaces over the code and press <RET>, the line will be deleted.

If the address is altered, then the program pointer will move to the new address provided that it is within the range of the program that already exists.

Labels: Address labels may be used followed by a colon:

```
eg.LABEL:NOP
         JP LABEL
```

Comments: Comments may be written after any instruction by preceding the comment with a semi-colon:

```
eg. RET;End of Prog
```

DS: DS may be used to define a block of space up to 254 bytes:

```
eg. DS 200.
```

DB: Bytes can be defined as a list of numbers or by enclosing characters within "":

```
eg. DB 1,2,"ABC"
```

WARNING When you exit from the assembler, all the code is assembled and all addresses are calculated. It is now possible to edit your BASIC program but if the assembly code is moved by inserting new lines beneath it, you must ensure that the address are still calculated correctly. To do this simply enter the assembler with each code line in turn and exit again, thereby reassembling each program.
It would be sensible to write your assembly code as the first few lines of the program if it is to be merged with BASIC as BASIC lines edited above do not affect those with lower line numbers.

Having written your program, you execute it by typing RUN <RET> as with BASIC.

FRONT PANEL
The FRONT PANEL is provided for you to test and debug your programs. Its effectiveness is dependent on the skill which you will aquire by discovering what it can do for you.

Type PANEL <RET>

You will see that the Z80 registers are displayed on the right and a block of memory at the bottom.

Type L2000

and a block of code will be listed. The panel will list programs, display memory and registers and allow you to test your programs by stepping through them one instruction at a time.

If an assembly program is written and run using the MTX assembler, the break key can be used to stop the program and the PANEL will display its current status.

See the reference section for the instructions for the PANEL.

ABS(<number>)

Gives the absolute value of the specified number. The result has the same magnitude but the sign will always be positive.

e.g.

```
ABS (59) = 59
ABS (-59)= 59
```

ADJSPR p,n,v

This command adjusts a value which has been assigned to a sprite either by the SPRITE command or subsequently by ADJSPR or MVSPR. The advantage of ADJSPR is that unlike SPRITE, only one parameter is altered at a time thereby increasing the speed of updating single values.

n is the sprite number.
v is the new value to be assigned to the parameter chosen by p.

p	meaning	range of v
0	pattern	0 to 31 (size 1) 0 to 127 size 0
1	colour	0 to 15
2	x pos	0 to 255
3	y pos	0 to 255
4	x speed	0 to 255 (128 to 255 = neg)
5	y speed	0 to 255 (" " ")

AND

See BOOLEAN EXPRESSIONS

ANGLE <angle>

The computer holds a 'direction' which is used in commands such as ARC, DRAW and MVSPR.
The direction is made up of two components...PHI and ANGLE

ANGLE Initialises the direction to the specified angle. The angle is in radians measured in an anticlockwise direction from the horizontal.-->

PHI <angle>

Each time a PHI command is executed, the 'direction' is adjusted by the specified angle.

e.g.
```
10 VS 4
20 CLS
30 ANGLE 0
40 FOR I=1 TO 20
50 PLOT 100,100
60 PHI .1
70 DRAW 50
80 NEXT
90 GOTO 90
```

For conversion to degrees... degrees=radians*2*PI/360

For conversion to radians... radians=degrees*360/2/PI

ARC <length>,<angle>

This command draws an arc of a circle. The starting position is the current plot position and the initial direction is the direction currently held by the computer. Both the plot position and the direction are updated. The angle parameter determines the curvature of the arc by specifying what angle is subtended. In other words the larger the angle, the tighter the curve. If the angle is greater than 360' or 2*pi radians, the arc will retrace its path.

ASC (<string>)

Gives the code of the first character of the string

e.g.

```
10 LET A=ASC("B")
20 PRINT A
```

Will print 66 which is the ASCII code of "B"

ASSEM <Line No>

Switches on the assembler to assemble at the specified BASIC line. (See CODE)
If the Line already exists, the assembler will only be entered if the specified line is a CODE line.

Refer to the Assembler section of the manual.

ATN (<number>)

Gives the angle whose tangent is the specified number.

ATTR p,state

ATTR determines the effect on the graphics screen of using one of the plotting commands such as PLOT, DRAW or ARC.
The attributes are not exclusive but may be used in any combination.

The state of the attribute is either on or off where

1=ON
0=OFF

p takes the value 0,1,2 or 3.

p=0 Inverse print. Characters are printed in the paper colour on the ink colour.

p=1 Over print. Characters are merged with those alread present.

p=2 Unplot. Plots the paper colour.

p=3 Over-plot. Plots the ink colour if paper was there before and the paper colour if ink was there before. During CLS, and other functions, text is not over written but colours may change.

The effect during plotting is to do nothing. This can be used to move the PLOT=SPRITE around the screen whilst leaving the screen unchanged.

AUTO <Line no>,<increment>

This command switches on the automatic line numbering.
The numbering will start at the specified line no and will be incremented by the specified increment.

e.g.

```
AUTO 10,10
```

This will result in lines being numbered 10,20,30,40...

To switch of the numbering press the CLS key followed by <ret>

AUTO SCROLL

The auto scroll facility is provided to allow the computer to automatically halt printing to the screen when the screen is full.
It can be switched on and off by the user or by the programmer.

The PAGE key is used as a switch to switch the AUTO SCROLL off and on.
If the AUTO SCROLL is switched on however, any key can be used to tell the computer to continue.

e.g. Type the following

```
10 FOR I= 1 TO 1000
20 PRINT I
30 NEXT

RUN
```

Press the PAGE key and the printing will stop.

Press any key once and the printing will continue for one more screen. This may be repeated any number of times.
Pressing the PAGE key again will turn off the AUTO SCROLL.

The PAGE key alone may be used as a switch to stop and start printing at will.

The programmer may control the scrolling using the escape sequence ESC P as a switch in the same way as the PAGE key.

For example:

```
10 PRINT CHR$(27);"P";
20 FOR I= 1 TO 1000
30 PRINT I
40 NEXT
```

In this case it will be seen that when the program is run the AUTO SCROLL will be switched on.

See LIST,ASSEM,PRINT.

BAUD <Channel>,< Baud rate >

Sets the RS232 channel 1 or 0 to the selected baud rate.
The following rates are allowed.

75	1200
110	2400
150	4800
300	9600
600	19200

e.g. BAUD 0,1200

BOOLEAN EXPRESSIONS

The computer needs a way to combine expressions logically to give one value which is either true or false so that a decision can be made according to a single value.

```
10 INPUT "ENTER X,Y ";X,Y
20 PRINT "X=1 ",X,(X=1)
30 PRINT "Y=2 ",Y,(Y=2)
40 PRINT "X=1 AND Y=2",(X=1 AND Y=2)
50 GOTO 10
```

Notice that when truth values such as y=1 are printed or used in expressions, they are enclosed by brackets.

When the above program is RUN enter values for X and Y and look at the results. You will see that if an expression is true the result is 0, otherwise it is -1. There are no other TRUTH values.

Expressions which yield a TRUTH value are called BOOLEAN expressions. When a Boolean expression is used in an IF statement, it need not be enclosed in brackets.

For example:

```
40 IF X=1 AND Y=2 THEN STOP
```

RULES FOR BOOLEAN EXPRESSIONS.

There are three BOOLEAN OPERATORS, AND OR and NOT.
There are 6 relational operators <,>,=,<>,<=,>=

A relational expression is a relation between two values of the same type.

For example:

```
X<>2
A$="AAA"+"BBB"
(X=2)=(Y=3)
```

Relational expressions yield truth values.

A Boolean expression is an expression which yields a truth value and so relational expressions are also Boolean expressions. However using the Boolean operators AND OR and NOT to combine relational expressions, more complex relationships between values can be evaluated.

For example:

```
10 PRINT (NOT 2=2)
20 PRINT (NOT 2=2 OR 3=2)
```

An example of the use of Boolean expressions is given below.

```
10 INPUT X,Y
20 IF X=2 AND Y=2 OR Y=7 THEN STOP
30 GOTO 10
```

This example will stop if either Y=7 or both X=2 and Y=2.

In an expression like

```
10 PRINT (2*2=5 OR 3+3=4 OR 2=2 AND 1=2)
```

we need to know in what order the expression is going to be evaluated.

Just as there are rules for evaluation of arithmetic expressions, there are also rules for evaluation of any type of expression. We know that * has a higher priority than + so that

```
3*4+5 = (3*4)+17   and not 3*(4+5)=27
```

To know how a complicated expression is going to be evaluated we follow a few simple rules.

Arithmetic operators have highest priority.
Relational operators all have the same priority which is less than all arithmetic and greater than AND OR and NOT.
AND, OR and NOT have the lowest priority with AND having the highest and NOT the lowest.

ORDER OF PRIORITY

```
^
*     /
+     -
=     <>    <     >     <=    >=
AND
OR
NOT
```

The priority defines the order of evaluation, which as with arithmetic can be altered by the use of brackets.

For example:

```
10 PRINT (2*2=5 OR 3+3=4 OR 2=2 AND 1=2)
```

is the same as

```
10 PRINT (  ((2*2)=5)  OR ((3+3)=4)  OR  ((2=2) AND (1=2))  )
```

CHR$ (<number>)

Gives the character whose code is the specified number.

For example:

```
PRINT CHR$(65)
```

Will print the character `A'.

CIRCLE X,Y,r

Draws a circle of radius r with centre X,Y

For example:

```
10 VS 4
20 CLS
30 CIRCLE 100,100,50
40 PAUSE 1000
```

CLEAR

CLEAR removes all of the variables .

CLOCK <String>

e.g. CLOCK "120000"

The clock is initialised to the value of the string.
The clock is a 100 hour clock which counts accurately in seconds minutes and hours up to 100 hours when it resets to 0.

To print the time see TIME$.

For example: To print the time in the top left corner:

```
10 CLOCK "120000"
20 CLS
30 PRINT TIME$;CHR$(26);
40 GOTO 30
50 REM CHR$(26) Homes the cursor.
```

See TIME$

CLS

Clear screen

In TEXT mode CLS will clear the screen (or Virtual screen).

In GRAPHICS mode CLS will clear the screen unless one of the screen attributes has been set using the ATTR function, in which case it may be necessary to switch off the attribute before CLS will operate.

CODE

Code is not a command but rather a word to indicate that the following lines are assembly language. The word can not be typed and is inserted by the assembler into the BASIC line.

For example:

```
10 REM START OF PROGRAM
20 CODE
8030 LABEL: LD A,(HL)
8031        RET

Symbols :   8030  LABEL

30 REM END OF PROGRAM
```

All labels are listed at the end of each block of code.
Labels are local to each block.

COLOUR p,n

The COLOUR command sets the colours for the graphics screen.
n selects the colour (see colour table in the appendix).

The value of p selects which areas of the screen the colour refers to.

Two sets of colours are defined. The print colours refer to colours that will be used when colours are printed. The non-print or PLOT colours refer to colours that will be used when colours are plotted or when screen functions are used.

p=0 Print paper.
p=1 print ink.
p=2 non-print (plot) paper.
p=3 non-print ink.
p=4 border colour.

CONT

CONT can be used directly after a STOP command or after pressing the break key to restart the program. Any editing or alteration of the program will prevent CONT from operating.

COS (<angle>)

Gives the cosine of the angle specified in radians.

CRVS n,t,x,y,w,h,s

To create your own virtual screen, use the CRVS command to define the required area and then select the screen using the VS command.

n	Virtual screen reference number. (0 to 7)
t	Type of screen. 0=text,1=graphics.
x	Coordinates of top left corner.
y	
w	Width of virtual screen in characters.
h	Height of screen in lines.
s	Width of screen. (40 for text, 32 for graphics)

If s is a different value, to the actual width of the screen, the virtual screen will become distorted. This can however be used to advantage. If for example s is set to 80 in a text screen, the virtual screen will only allow printing on alterate lines.

CSR X,Y

Moves the cursor to position X,Y

e.g.

```
10 CSR 12,10
```

Moves the cursor to position 12,10.
Any subsequent Printing or Input will occur at the new cursor position.

CTLSPR p,x

p = parameter and can be any of the six below:

0 Speed
1 to 255 to 0 (1 is fastest)

1 Distance
Tells the computer to move the sprite by `x' pixels when requested.

2 Number of sprites
0 to 32 (The number of sprites must be at least 1)

3 Number of circling sprites
Sprites that will orbit when they go off the edge of the screen (must not exceed number of sprites)

4 Plot sprite
A PLOT SPRITE can be chosen which will subsequently appear whenever a point is plotted. This sprite will move around the screen following any points or lines drawn by the BASIC GRAPHICS commands. This sprite can be any of the 32 defined in the normal way.

5 number of moving sprites 0 to 32
This is the number of sprites that will move by themselves according to the x-speed and y-speed set in the SPRITE and ADJSPR commands.

6 magnitude and size

x=0	size 8X8	mag 1
x=1	size 8X8	mag 2
x=2	size 16X16	mag 1
x=3	size 16X16	mag 2

DATA <list of values>

If the computer encounters a READ command the program is searched for the first DATA statement. Once a DATA statement has been found a pointer to the values in the DATA statement is maintained. Each time a READ statement requires a value, the value at the DATA POINTER is assigned and the pointer is moved to the next value.

For example:

```
10 REM PROGRAM TO PRINT FROM A DATA STATEMENT
20 FOR I=1 TO 10
30 READ X
40 PRINT X
50 NEXT I
60 DATA 1,1,2,2,3,3
70 DATA 4,4,5,5
```

DATA statements aren't themselves executed and will be ignored other than when required by a READ statement.

If all of the data on a line has been read by a READ statement, the computer will search for the next DATA statement and update the DATA POINTER accordingly.

If no more DATA is available, a No Data error will occur.
You may find that the space after the word DATA is accepted as part of the string when READing strings. To guarantee that this does not happen, the abbreviated form DAT. can be used.

e.g.

Type 10 DAT.AAA,BBB

instead of

10 DATA AAA,BBB

See READ and RESTORE

DIM <array list>

Before an array can be used, space must be made in the computer memory using a DIM statement.

For example:

```
10 DIM A(10,10),A$(1000)
```

Each element of a numeric array takes 5 bytes and each element of a character array takes 1 byte.

An array cannot be redimensioned unless all of the variables are cleared.

For example:

```
10 DIM A(10)
20 DIM A(20)
```

This will result in an error as the array already exists by the time the program reaches line 20.

e.g.

```
10 LET A$="ABC"
20 DIM A$(100)
```

This will result in an error since line 10 automatically defines a character array large enough to hold the data "ABC". In fact, the computer will make space enough for A$ to hold 64 characters.

All strings are treated as character arrays rather than variable length strings. The amount of space allocated to the character array will depend on whether or not a DIM statement is used.

If a DIM statement is used, the exact number of characters specified will be allocated.

For example:

```
10 DIM A$(1)
```

This will result in a string of length 1 character

If a string is used without a DIM statement, the space allocated will depend on the first assignment where the amount of space will be the amount required increased to the next multiple of 64.

For example:

```
10 LET A$="ABC"          Space = 64 bytes
20 LET B$(100)="X"       Space = 128 bytes
```

DRAW X

Draws a line of length X from the current plot position in the current direction. The plot position is updated to the end of the line.

DSI

Direct Screen Input

This command allows you to roam about freely within a screen only ending when carriage return is pressed. Within this instruction, the break key is not operational but will generate CTL C.

```
CTL W = tab back
CTL ] = PMODE
CTL \ = SMODE
CTL ^ = CURSOR ON
CTL _ = CURSOR OFF
CTL D   letter A to O = paper A to O (1 to 15)
CTL F     "       "       ink A to O (1 to 15)
ESC I = insert line
ESC J = delete line
ESC K = duplicate line
```

EDIT <Line no>

This command causes the specified line to be copied into the EDIT screen for editing.

For example:

```
10 REM ABCDEDFG

EDIT 10
```

If the line number is changed a copy of the line will be created at the new line number.

To edit a CODE line, the assembler should be entered and the assembler editor invoked. (see the assembler section of the manual)

EDITOR <variable list>

The editor gives the programmer the facility to accept input from a defined area of the screen. The area is defined by virtual screen 0 which may be set using the CRVS command.

For example:

```
10 CRVS 0,0,20,10,10,1,40
20 EDITOR A$
30 VS 5
40 PRINT A$
50 GOTO 20
```

The EDITOR leaves the current screen as screen 0 and must therefore be reset if printing is required on the full BASIC screen or any other screen. Line 30 resets the current screen to the full basic screen.

ELSE

See IF.

EXP (<number>)

EXP is the exponential function whose value is e raised to the power of the specified number.

For example:

```
EXP (1) = 2.71828183
```

FOR <control variable> = <start> TO <limit> {STEP <increment>}

NEXT <control variable>

<control variable> is a simple numeric variable but in this instance is called the control variable.

<start>, <limit> and <increment> are numeric expressions.
If STEP <increment> is not present then the computer will behave as if STEP 1 were present.

FOR and NEXT delimit a block of program.

For example:

```
10 FOR I= 0 TO 10 STEP 1
20 PRINT I,I*I
30 NEXT I
```

When the computer meets the FOR statement in a program, it assigns <start> to the <variable> just as if it were a LET statement.

Execution of the program now continues at the following line until a NEXT statement is encountered with the same control variable. At this point <increment> is added to the control variable and the new value of the control variable is compared with the <Limit>. If the limit has been reached, the program continues after the NEXT statement otherwise control is returned to the statement following the FOR statement.

The increment may be negative.
If the control variable is not present in the NEXT statement, an appropriate variable will be assumed.

See NEXT

GENPAT p,n,d1,d2,d3,d4,d5,d6,d7,d8

The GENPAT command is the command used to generate all types of patterns required by BASIC for characters and SPRITES. There are 5 modes.

1. To redefine an ASCII character. (CODES 32 TO 127)
2. To define a non ASCII character. (CODES 129 TO 154)
3. To define colour for each line of a character.
   This only applies to user definable characters with codes 147 to 154.
4. To define an 8 by 8 sprite pattern.
5. To define each quadrant of a 16 by 16 sprite.

User definable characters have codes from 129 to 154.

Mode 1 allows the user to redefine one of the standard ASCII character patterns. Note that the ASCII characters are the ones which are most often used by the computer

Mode 2 allows the user to define his own character patterns without destroying any of the standard ascii characters.

Mode 3 allows some of these user defineable characters to be further defined by specifying an ink and paper colour for each of the eight rows of the character.

The values for ink and paper are as specified in the colour table in the appendix but in this instance we are specifying two colours (ink and paper) at the same time. Each of d1 to d8 specify a paper and ink colour as a single number:

```
          bit  0 1 2 3: 4 5 6 7
               ink    : paper

        value =  16   *   paper + ink
```

e.g. Red ink on blue paper

```
          =      RED : BLUE

          =  16   *   BLUE + INK

          =  16   *   4    + 9

          =  73
```

MODE	P	N	
1	0	ascii code (32 to 127)	
2	1	user definable (code 129 to 154)	
3	2		
4	3	pattern number	8 by 8 sprite pattern
5	4	pattern number	16 by 16 NW quarter
	5	pattern number	16 by 16 SW quarter
	6	pattern number	16 by 16 NE quarter
	7	pattern number	16 by 16 SE quarter

GOSUB <line no>

Continue execution from the specified line until a RETURN statement is encountered at which point return control to the statement following the GOSUB

For example:

```
10 FOR I= 1 TO 10
20 GOSUB 100
30 NEXT
40 STOP
100 PRINT  I,SIN(I),COS(I)
110 RETURN
```

Gosubs may be nested up to 34 deep at which point an error will occur.

See RETURN.

GOTO <Line no>

Control is passed to the BASIC line specified.

For example:

```
10  GOTO 40
20  PRINT "LINE 20"
30  STOP
40  PRINT "LINE 40"
```

If the line doesn't exist an error will occur.

GR$ x,y,b

GR$ reads a bit pattern from a graphics screen, returning the value as a character. This function should be used if you wish to print the graphics screen to a high resolution printer.

x and y are locations on the virtual screen.

b is the number of bits to be read (if b = 1 equivalent to "POINT" function).

The bits are read in a vertical direction. ie. GR$(20,190,4) gives a character made up as follows:

```
bit 7       0
bit 6       0
bit 5       0
bit 4       0
bit 3       pixel at 20,190
bit 2       pixel at 20,189
bit 1       pixel at 20,188
bit 0       pixel at 20,187
```

IF <boolean expression> THEN <statement> {ELSE <statement>}

The IF statement allows the program to branch depending on whether a condition is true or false.

<boolean expression> is any expression which yields a truth value.

e.g. Y=2 AND X=3

<Statement> is any legal BASIC statement (which may include an IF statement).

If the boolean expression is true, the statement after the THEN is executed.

If the boolean expression is false and ELSE is not present, the program continues at the next line.

If the boolean expression is false and ELSE is present, the program continues after the ELSE.

For example:

```
10 INPUT "ENTER Y OR N ";A$
20 IF A$="Y" THEN GOTO 40 ELSE GOTO 100
30 GOTO 10
40 PRINT "YES"
50 STOP
100 PRINT "NO"
```

See BOOLEAN EXPRESSIONS

INK <colour>

Selects the INK colour. <colour> is a number in the range 0 to 15 and selects a colour from those in the colour table in the appendix.

INPUT {"string";}<variable list>

The INPUT command is used to input information into the computer. The variable list is a list of array or numeric variables separated by commas.

e.g.

```
INPUT A,B,ABC$
```

If the {string} is not present, a question mark will appear as a prompt whenever the INPUT statement is used.
If non-numeric information is entered into a numeric variable, or the too few items are input, a question mark will appear after information which means that it should all be typed in again.

If the {string} is present, it replaces the question mark as a prompt. The string must be followed by a semicolon.

e.g.

```
INPUT "ENTER YOUR NAME";N$
```

INT (<number>)

Gives the integer part of the number.

For example:

```
INT  (2.5) = 2
INT (-2.5) =-2
```

LEFT$(<string>,<number>)

The string is truncated after the specified number of characters.

For example:

```
LEFT$("abcdef",3) = "abc"
```

LEN <SPACE>(<string>)

Gives the length of a string.

Notice that a space must be left between LEN and (<string>) otherwise LEN will be considered as a numeric array.

For example:

```
PRINT LEN ("ABC"+"DEF")
```

Will print 6.

LET <variable>=<value>

The LET statement assigns a value to a variable.

For example:

```
10 LET X=2
20 LET A$="abc"
```

This assigns the value 2 to the variable X and the value "abc" to the variable A$.

The value and variable must be of the same type.
i.e. numbers cannot be assigned to string variables and strings cannot be assigned to numeric variables.

LIST {<start line no>}, {<finish line no>}

LIST lists a program to the screen.

There are three different formats depending on how many line numbers are specified.

LIST
Lists the entire program from start to finish.

LIST 100
Lists the program from line 100 to finish.

LIST 100,200
Lists the program from line 100 to line 200.

LIST 100,100
Lists line 100 only.

See LLIST, AUTO SCROLL

LLIST

LLIST lists a program to the printer.

The formats are as for LIST

See LIST.AUTO SCROLL

LN (<number>)

Gives the natural log of the specified number.

LPRINT [List of expressions]

LPRINT has exactly the same format as PRINT but sends output to the printer instead of the screen.

See PRINT.

MANIPULATING STRINGS

An MTX string is treated as a character array as if a DIM statement had been used to make space for it.

e.g.

```
10 LET A$="AAA"
```

is equivalent to

```
10 DIM A$(64)
20 LET A$="AAA"
```

The MTX allows selection of parts of a string by use of one fewer or one more subscript than are normally required.

Since a string is considered as a one dimensional character array, specifying a single subscript would refer to a single character at the subscript position.

e.g.

```
10 LET A$="ABCDEFG"
20 PRINT A$(3)
30 PRINT A$
40 PRINT A$(3,3)
```

Line 20 will print the letter 'C'

If one subscript too few is specified as in line 30, the entire string will be printed.

If one subscript too many is present the part of the string will be printed starting at the first subscript for as many characters as are specified in the second subscript.

Line 40 therefore will print 'CDE'

These rules can be extended for character arrays with any number of dimensions.

MID$(<string>,X,Y)

Gives Y characters starting at position X in the string.

For example:

MID$("ABCD"+"EFGH",3,4) = "CDEF

MOD (X,Y)

Gives the remainder on dividing X by Y.

e.g. MOD (10,7)=3

MOD (X,Y) is equivalent to X-INT(X/Y)*Y

MVSPR p,n,d

MVSPR is a general purpose command which combines 4 distinct functions

p	meaning
1	MOVEMENT
2	PATTERN SELECTION
4	REDIRECT
8	PLOT AT CENTRE

The functions are combined to allow complicated movements to occur whilst using only a single instruction. The type of activity is selected by p as in the table above. If combinations of activities are required, just add the p values together. Some examples are given below.

		eg 1	eg 2	eg 3
MOVE	1	YES	YES	YES
PATTERN	2	NO	YES	NO
REDIRECT	4	NO	YES	YES
PLOT AT CENTRE	8	YES	NO	YES
TOTAL p value		9	7	13

n selects the sprite number.

d is slightly more complicated than with other graphics commands as it must be able to reflect a value for several activities. If d is not in the range of any one of the chosen activities an error will occur.

MOVE (p=1) moves the sprite 1 step in the direction specified by d. The step size is set in CTLSPR 1 and the direction must be in the range 0 to 8 where directions 0 and 8 bare the same.

PATTERN changes the sprite pattern to pattern number d. This pattern should have been defined in a GENPAT statement.

REDIRECT picks up current velocity vector and switches it to the new direction.

PLOT AT CENTRE causes a point to be plotted at the centre of the sprite specified by n.This is not directly affected by the value of d at all.

NEW

This command resets the computers system variables thereby preparing it to accept a new program.

NEXT <Control variable>

Next specifies the end of a FOR statement block.

If the control variable is specified, the NEXT is matched with a FOR and all nested FOR blocks which are either complete or incomplete are terminated.
If the variable is not specified it is assumed that this NEXT belongs to the last active FOR statement.

See FOR.

NODDY

Pass control to the Noddy editor.
Noddy should appear at the base of the screen.

Noddy commands:

A		Advance to next program page on the stack.
B	label	Branch to a label.
D	page	Display a Noddy page (on virtual screen 5)
E		Enter input (into virtual screen 7)
G	page,{label}	Goto page at label if specified
I	match,label	If input = match then goto label
L	page	List a Noddy page to the printer
O		Remove a program page from the stack
P		Pause before continuing with the program
R		Return to BASIC
S	page,page...	Stack up program pages

The Noddy interpreter accepts its input from virtual screen 7 and displays on virtual screen 5.
These virtual screens will normally be the complete screen for display and the bottom line for input but they may be redefined using the CRVS command.

e.g.

```
10 CRVS 5,0,10,10,20,5,40
20 CRVS 7,0,10,16,20,1,40
30 NODDY
RUN
```

Two smaller screens should be defined for use by Noddy leaving everything else unchanged.

See Noddy Section of the manual.

NOT

See BOOLEAN EXPRESSIONS

ON GOTO and ON GOSUB

The ON command is used when you want to GOTO or GOSUB to a part of the program depending on the value of a variable.

e.g.

```
10 LET X=2
20 ON X GOTO 100,200,300,400,500
```

This part of a program would goto line 100,200 etc depending on the value of X.

If X=0 it would branch to 100, X=1 would branch to 200 ....

Similarly, in the example below, the program would go to the subroutine at 100,200,300,400 or 500 depending on the value of X and return to line 30 when a RETURN is encountered.

```
10 INPUT X
20 IN X GOSUB 100,200,300,400,500
30 REM Program continues here.
```

OR

See BOOLEAN EXPRESSIONS

OUT <port>,<value>

Outputs the specified value to the specified port.
Refer to Technical Manual - System Block Diagram

PANEL

Switch on the front panel.

Basic	Exit?	Answer Y to return to BASIC
Clear		Clears the List screen
Display	Hex	Display a block of memory around Hex.
Go	Hex1 to Hex2	Run a program starting at Hex1 upto Hex2.
I		Display ASCII/HEX
List	Hex	List from Hex L. lists from the program counter
Move Hex1	End Hex2 To Hex3	Moves a block of memory Hex1-Hex2 to Hex3.
Register	Hex	Change register at register cursor To Hex
Single step		Execute the command at the program counter
Trace		As S but calls are treated as a single instruction.
X		Display the alternate register set.
.		Move the register cursor.
-		Move the display cursor back.
<ret>		Move the display cursor forwards.
▲(cursor up)		Move display cursor up.
▼(cursor down)		Move display cursor down

Example of PANEL Screen

```
0100          JP Z,E8
0103          LD A,0C3        AF >0000 C3
0105          LD (0050),A     BC  0000 C3
0108          LD (0053),A     DE  0000 C3
010B          LD HL,2B09      HL  0000 C3
010E          LD (0051),HL    IX  0000 C3
0111          LD HL,3B04      IY  0000 C3
0114          LD (0054),HL    SP  0100 C3
0117          LD A,0E1        PC  0100 C3
0119          LD (0028),A
011C          LD A,7E
011E          LD (0029),A
0121          LD A,0FE
0123          LD (0020),A
```

JP 02E8

```
00F0: 21 2A 87 28 A9 A9 A7 2F
00F8: D0 02 5C 4D A7 2F 9D 12
0100: C3>E8 02 3E C3 32 50 00
0108: 32 53 00 21 09 2B 22 51
0110: 21 2A 87 28 A9 A9 A7 2F
0118: D0 02 5C 4D A7 2F 9D 12
```

PAPER <colour>

Selects the paper colour. <colour> is a number in the range 0 to 15 and selects a colour from those in the colour table in the appendix.

PAUSE <number>

The program will pause for a length of time dependent on number. The time cannot be specified accurately but will vary according to how many incidental functions the computer is performing at the same time such as flashing the cursor or updating the clock.

PEEK <address>

Gives the contents of the specified address in the current page.

Notice that the MTX pages its memory in 32K blocks and so care must be taken to ensure that when PEEK is used you are peeking the correct page. The top 16K of memory is available to all pages.

PHI <angle>

See ANGLE.

PI

The MTX stores an accurate value of PI so that it doesn't have to be calculated each time that it is needed. PI can be considered as a number and used whenever a number would be used.

e.g.

```
PRINT PI
PRINT COS(PI/2)
```

PLOD "string"

PLOD is used to run a Noddy program where string is the name of the Noddy page at which execution starts.

e.g

10 PLOD "PROG1"

This will run a Noddy program starting with the page called PROG1.

PLOD may be used within a BASIC program and control may be switched backwards and forwards. When a Noddy program returns, execution continues at the next basic line.

PLOT x,y

Plots a point in the graphics screen at the point x,y.

See COLOUR and ATTR

POKE <location>,<value>

The POKE command loads a the specified memory location with the specified value.
If more than 32K of memory is present, the basic program may spread over several pages (see the ROM BASED MEMORY MAP).The top 16K will always be present but POKing into any other area may be dangerous unless other precautions are taken.

e.g.

POKE 50000,100

Loads 100 into location 50000

PRINT <expression list>

The print command is used to print information onto the television screen or monitor.

e.g.

```
PRINT "SIN OF 8 = ";SIN(8)
```

PRINT is followed by a list of expressions separated by commas or semicolons. The expressions can be string expressions or mathematical expressions and any number of commas can be used to tabulate the information.

A comma moves the cursor to the next TAB position.

A Semicolon leaves the cursor immediately after the printed information.

RAND <number>

This command sets the seed for the random number generator. If no number is specified the seed will be set to a random number.

e.g

```
10 RAND 1000
20 FOR I=1 TO 100
30 PRINT INT(RND*50),
40 NEXT I
```

See RND

READ <variable list>

e.g.

```
10 READ A,B,A$,B$(8)
20 PRINT A,B,A$,B$
30 DATA 1,2,AA,BB
```

Values are read from the DATA statements (See DATA) sequentially into the variables specified in the READ statement.

If an invalid assignment is made to a numeric variable an error will occur.

Data statements may occur anywhere within a program and are accessed sequentially in order either from the start of the program or from the value specified in a RESTORE statement.

See DATA,RESTORE

REM <anything>

The REM command allows comments to be inserted into a program. The REM statement is ignored by the computer and is used only for documentation.

RESTORE <line no>

Restore tells the computer from which line the next READ statement should start reading the next DATA item
This is particularly useful if the same DATA has to be read in two or more parts of a program.

e.g.

```
10 DIM A(20)
20 FOR I= 1 TO 10
30 READ A
40 PRINT A
50 NEXT I
60 RESTORE 100
70 FOR I= 1 TO 10
80 READ A(I)
90 NEXT I
100 DATA 1,2,3,4,5,6,7,8,9,0
```

See DATA,READ

RETURN

Returns control to the line following the last GOSUB executed
If no GOSUB has been executed, an error will occur.

See GOSUB

RIGHT$(<string>,<number>)

The specified string is truncated on the left leaving the specified number of characters on the right.

e.g.

```
RIGHT$("ABCDEFG",3) = "EFG"
```

RND

RND returns a pseudo random number.

See Rand

ROM <rom number>

Passes control to an additional ROM pack for example PASCAL or FORTH. Details will be supplied with the ROM

It is dangerous to use this command unless the appropriate ROM is actually present.

RUN

RUN tells the computer to start running the program from the beginning.
All variables will be cleared.
A program can also be run by using a GOTO statement as a direct command. In this case the variables will be unaltered.

e.g.

```
10 REM START
20 PRINT 1,2,3
30 REM END

GOTO 20
```

SBUF <Number>

This command makes space in a sound buffer for use by the SOUND command.

e.g.

```
10 SBUF 8
```

This will make space for 8 blocks of sound data for each of the 3 channels and the noise channel.
Each block takes 10 bytes. The above statement therefore takes a total of 320 bytes. i.e. 8 * 4 * 10.(8*channels*block length).
The Buffer is made at the top of memory below the system variables

SGN (<number>)

Gives a result depending on the sign if the specified number.

If the number is positive the result is +1
If the number is negative the result is -1
If the number is zero the result is 0.

e.g.

```
SGN(-2.5)= -1
SGN(2.5) =  1
SGN( 0 ) =  0
```

SIN (<angle>)

Gives the sine of the angle specified in radians.

SOUND <expression list>

The effect of the sound command depends on the number of expressions in the expression list.

SOUND <channel>,<frequency>,<volume>

(3 parameters)

Channel = 0,1,2 or 3
0,1 and 2 are pure sound channels.
3 is the noise channel.

SOUND <channel>,<freq>,<vol>,<freq inc>,<vol inc>,<duration>,<mode>

(7 parameters)

Channel,freq and volume are as described above.
Every 1/64th of a second the computer adds the frequency increment to the frequency and the volume increment to the volume. This continues for a length of time equal to the duration which is also measured in 1/64ths of a second.

The mode can be either 0 or 1

If mode=0 the freq and vol parameters will be ignored which means that the increments will increment whatever values of frequency and volume were current when the command was encountered.
If mode=1 the freq and vol will be loaded into the sound buffer to initialise the frequency and volume of the relevant channel.

SPK$ (screen peek)

Peeks the character at the cursor location and auto increments the cursor location. The character is returned as ASCII.

e.g. To read characters from the screen into an array, and reprint them.

```
10 CLS
20 FOR I=32 TO 64
30 PRINT CHR$(I);
40 NEXT
50 CSR 0,0
60 FOR I=32 TO 64
70 LET A$(I)=SPK$
80 NEXT
90 PRINT
100 PRINT A$
```

SPRITE n,pat,xp,yp,xs,ys,col

n is sprite number 1 to 32

pat is pattern number 0 to 127 (size 0)
0 to 31 (size 1)

xp is position x off centre

yp is position y off centre (in range -4095 to 4095)
0,0 is defined as bottom left hand corner of screen i.e same as for plot.

N B Sprite coordinates are absolute and do not look at virtual screen origins (i.e assume a 32 by 24 graphic screen)

xs is the speed in the x direction range -128 to 127 where 1 unit of speed moves the sprite 1/8 pixel every master speed cycle as set by CTLSPR 0

ys is the speed in the y direction (plus = upwards) range -128 to 127

col is the sprite colour, 0 to 15

STEP

See FOR.

STOP

Stops execution of the program.
CONT may be used to continue execution provided that the program has not been altered in any way.

e.g.

```
10 REM LONG PAUSE PROGRAM
20 CLOCK "000000"
30 PRINT "START"
40 IF TIME$="000130" THEN STOP
50 GOTO 40
```

STR$(<number>)

Gives the string which represents the specified number.

e.g.

STR$(2+2) = "4"

Note that STR$ has a string value and can therefore not be used in numeric expressions.

TAN (<angle>)

Gives the tangent of the angle specified in radians.

THEN

See IF.

TIME$

Gives the time on the real time clock in the format

H H M M S S

Where H H is the number of hours elapsed since the clock was started with the CLOCK command. The hours will count up to 99 before resetting.

M M is the number of minutes.

S S is the number of seconds.

For example:

```
10 CLOCK "000000"
20 CSR 10,10
30 PRINT TIME$
40 GOTO 20
```

See CLOCK

TO

See FOR.

USR (<address>)

USR causes control of the program to be transferred to the specified memory address. This is the usual way of interfacing machine code to BASIC programs although the MTX assembler makes this function redundant in most cases.

On return to BASIC, USR has the value in the register pair BC.

e.g. if you assemble the following program and run it, '100' will be printed on the screen.

```
10 CODE
8007        LD BC,100
800A        RET

Symbols:

20 PRINT USR(32775)
```

32775 is the decimal of the HEX value 8007.

VAL (<string>)

Gives the numeric value of the specified string.
If the string is not a valid number, VAL will try to evaluate the string from the left hand side as a number until it can go no further.

For example:

```
VAL ("100")=100
VAL ("100000000000")=1E+12
VAL ("1000ABCDEF00")=1000
VAL ("12"+"12")=1212
```

VAL returns a number which may be used in arithmetic.

For example:

```
VAL("23")+10 = 33
```

VERIFY <string>

Verifies a program in tape against the program currently in the computer.

VIEW direction, distance

The Graphics screen can be considered as being a window into the sprite planes. The graphics screen is initially located as in the diagram below.

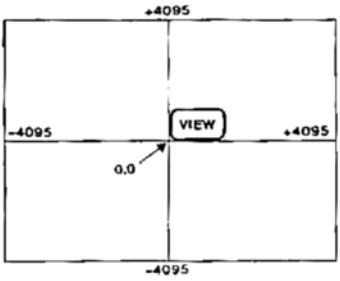
The view command moves the window relative to the sprite planes whilst leaving the position of the sprites unchanged.

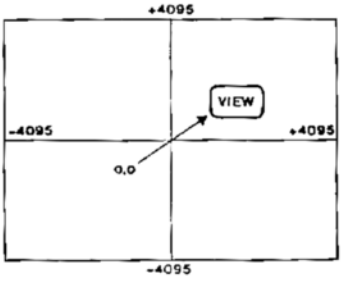
direction = 0 to 7
distance = 0 to 255

VS n

This command selects a virtual screen from those already defined or created using the CRVS command. The computer will automatically switch to the type of screen selected whether graphics or text.

See CRVS





SOFTWARE APPENDICES

1 ASCII Code Table
2 Control and Escape Sequences
3 Error Messages
4 Numeric Keypad
5 System Variables
6 Function Keys
7 Colour Table
8 Sound Tables
9 Absolute Directions
10 Flowchart conventions

APPENDIX 1

ASCII CODES

ASCII	HEX	DEC	ASCII	HEX	DEC	ASCII	HEX	DEC
NUL	00	0	/	2F	47		5E	94
SOH	01	1	0	30	48		5F	95
STX	02	2	1	31	49		60	96
ETX	03	3	2	32	50	a	61	97
EOT	04	4	3	33	51	b	62	98
ENQ	05	5	4	34	52	c	63	99
ACK	06	6	5	35	53	d	64	100
BEL	07	7	6	36	54	e	65	101
BS	08	8	7	37	55	f	66	102
HT	09	9	8	38	56	g	67	103
LF	0A	10	9	39	57	h	68	104
VT	0B	11	:	3A	58	i	69	105
FF	0C	12	;	3B	59	j	6A	106
CR	0D	13	<	3C	60	k	6B	107
SO	0E	14	=	3D	61	l	6C	108
SI	0F	15	>	3E	62	m	6D	109
DLE	10	16	?	3F	63	n	6E	110
DC1	11	17	@	40	64	o	6F	111
DC2	12	18	A	41	65	p	70	112
DC3	13	19	B	42	66	q	71	113
DC4	14	20	C	43	67	r	72	114
NAK	15	21	D	44	68	s	73	115
SYN	16	22	E	45	69	t	74	116
ETB	17	23	F	46	70	u	75	117
CAN	18	24	G	47	71	v	76	118
EM	19	25	H	48	72	w	77	119
SUB	1A	26	I	49	73	x	78	120
ESC	1B	27	J	4A	74	y	79	121
FS	1C	28	K	4B	75	z	7A	122
GS	1D	29	L	4C	76		7B	123
RS	1E	30	M	4D	77		7C	124
US	1F	31	N	4E	78		7D	125
space	20	32	O	4F	79		7E	126
!	21	33	P	50	80	DEL	7F	127
"	22	34	Q	51	81			
£	23	35	R	52	82			
$	24	36	S	53	83			
%	25	37	T	54	84			
&	26	38	U	55	85			
'	27	39	V	56	86			
(	28	40	W	57	87			
)	29	41	X	58	88			
*	2A	42	Y	59	89			
+	2B	43	Z	5A	90			
,	2C	44		5B	91			
-	2D	45		5C	92			
.	2E	46		5D	93			

APPENDIX 2

SOME USEFUL CONTROL AND ESCAPE SEQUENCES

CONTROL SEQUENCES

CTL Dn	Sets background colour to n
CTL E	Erase to end of line
CTL Fn	Sets foreground colour to n
CTL G	Sounds the bell
CTL H	Backspace, cursor left
CTL I	Tabulate the next block of eight columns
CTL J	Line feed, cursor down
CTL K	Cursor up
CTL L	Clear screen and home cursor
CTL M	Carriage return, cursor to left edge of screen
CTL W	Tab back
CTL Y	Cursor forwards
CTL Z	Homes cursor
CTL]	Page mode
CTL \	Scroll mode
CTL ^	Cursor on
CTL _	Cursor off

ESCAPE SEQUENCES

ESC S	Standard character font
ESC B0	American character font
ESC B1	English character font
ESC B2	French character font
ESC B3	German character font
ESC B4	Swedish character font
ESC B5	Spanish character font
ESC I	Inserts a blank line at cursor line
ESC J	Deletes the current cursor line
ESC K	Duplicates a line
ESC Xc	Simulates CONTROL character c

APPENDIX 3

ERROR MESSAGES

Params
Incorrect or wrong number of parameters for a function or command.

Mistake
A mistake has been made which should be obvious from the context.

A
Not outside virtual screen.

SE.A
Screen type not in type table.

SE.B
Invalid ESC sequence.

SE.C
Command not valid for this device.

SE.D
Switch to absent Virtual Screen.

SE.E
Invalid UDG/UDG type.

Symbol?
A symbol is missing, such as "=", "TO", "THEN", ","

Not numeric
A number is expected.

Not a string
A string is expected.

Boolean?
A truth value is expected.

Mismatch
An illegal relationship between different types of values.

BK
Break in tape LOAD or SAVE.

No data
No data for READ or No page for NODDY.

Overflow
Number too big.

Div /0
Division by zero

Out of range
Number is not in a valid range.

No space
To define an array
To expand a program
To assign a string to a character array
To perform a large operation.

Subscript
A Subscript is out of range or there are too many.

Gosub
Too many GOSUBS (more than 34).

Undefined
A variable is being used before it exists.

Array exists
An array has already been defined.

No FOR
A next has been encountered without a matching FOR.

No call
A RETURN has been encountered without a matching GOSUB.

No line
A reference is made to a non-existent line.

APPENDIX 4

THE NUMERIC KEYPAD

The Numeric Keypad has been designed for use with application programs. Notice that the Break key is in the top right hand corner of the numeric pad and you must decide if you want to allow this key to Break in or not. To use the numeric pad, press the shift key and one of the numbers. In this mode the Break/9 key will give a 9 and not Break.
There is however a numeric padlock which is set by a Bit in the keyboard flags. If this bit is set, the number pad will be locked to Numbers but for safety the Break key will over-ride the 9. To use the 9 you must turn off the Break key, by switching the Break key bit in the Interrupt flags INTFFF.

e.g.

```
10 POKE 64145,132
20 POKE 64862,13
30 PRINT INKEY$:GOTO 30
```

APPENDIX 5

SYSTEM VARIABLES

FA52	CTRBADR	Control buffers for sound
FA7A	STKTOP	Top of stack
FA7A	LSTPG	Number of 32K pages
FA7B	VARNAM	Bottom of variable names
FA7D	VALBOT	Bottom of variable values
FA7F	CALCBOT	Bottom of calculator stack
FA81	CALCST	Top of calculator stack
FA83	KBDBUF	Address of keyboard buffer
FA85		Syntax for user routine
FA89	USER	Basic user jump
FA8C		
FA8F	IOPL	List device
FA90	REALBY	Panel breakpoint
FA91	KBFLAG	See below
FA92	STKLIM	Top of free space
FA94	SYSTOP	Top of variables to be saved
FA96	SSTACK	Address of machine stack
FA98	USERINT	See below
FA9B	NODLOC	
FA9E	FEXPAND	Panel expansion
FAA1	USERNOD	Noddy expansion
FAA4	NBTOP	Top of Noddy
FAA6	NBTPG	Top of Noddy
FAA7	BASTOP	Top of current Basic page
FAA9	BASTPG	
FAAA	BASBOT	Bottom of Basic
FAAC	BASTPO	Top of each Basic page
FACC	ARRTOP	Top of arrays
FACF	BASELIN	
FAD1	BASLNP	
FAD2	PAGE	Current page configuration
FAD3	CRNTPG	Current Basic page
FAD4	PGN1	
FAD5	PGN2	
FAD6	PGTOP	
FAD8	GOSTACK	GOSUB stack
FB41	GOPTR	
FB43	GOSNUM	Number of nested GOSUBS
FB45	CTYSLT	Keyboard configuration
FB46	DATAAD	Data pointer
FB48	DATAPG	
FB49	DESAVE	

** System variables saved to here on tape **

```
FB4B    START       Keyboard buffer
FD48    STACK       Machine stack
FD48    SETCALL
FD4B    RICHJL
FD4E    USRRST      Restart 38
FD51    USERIO      See below
FD54    USERROR     Error trap
FD57    CLOCK       Real Time Clock
FD5E    INTFFF      See below
FD5F    CASBAUD     Cassette Baud rate
FD60    MIDVAL
FD61    RETSAVE     Start address for Auto Load
FD65    VAZERO
FD67    VERIF
FD68    TYPE
FD69    CONTFLG
FD6A    CONTAD }    Address of line to
FD6C    CONTPG }    confine after STOP or Break
FD6D    ASTACK
FD6F    TMPHL
FD71    TMPA
FD73    STACCT
FD75    PRORPL      See below
FD76    IOPR        See below
FD77    AUTOIN      Increment for Auto Line
FD79    AUTOST
FD7B    AUTOCT
FD7C    LASTKY      Last key pressed
FD7D    LASTASC     ASCII of last key read
FD7E    LASTDR
FD7F    RNSEED
FD81    BREAK
FD82    COMMAND     Address of First Command executed
FD84    ERRPOS      Position of syntax error
FD86    FLAGS1
FD87    ITYPE       Used by Assembler and Panel
FD89    MAFD
FD8B    MBCD
FD8D    MDED
FD8F    MHLD
FD91    MAF         }
FD93    MBC         }
FD95    MDE         }     Temporary locations for
FD97    MHL         }     storing registers
FD99    MIX         }
FD9B    MIY         }
FD9D    MSP         }
FD9F    MPC         }
FDA1    MEMPOINT
FDA3    WCHJUMP
FDA5    POINTERR
FDA6    DADD
FDA8    ASBYTE
FDA8    INDEX
FDAA    DBYTE
```

```
FDAC   LINKER
FDAD   EDIT
FDAE   LENGTH
FDAF   DETYPE
FDB0   DTYPE
FDB1   DISAD
FDB3   DPROG
FDB5   LABTABL
FDB7   APROG
FDB9   ENDTAB
FDBB   COMMENT
FDBC   COMAD
FDBE   ADLABEL
FDC1   INDEXLAB
FDC3   DATALAB
FDC6   DBLABEL
FDC7   BASEM
FDC9   CURLAB
FDCC   ACC1        Used by Maths
FDF2   INTTAB      Node Interrupt Table
FE02   GASH        Used by Sound
FE04   TEMP
FE14   CHAN
FE16   FREQ
FE18   VOL
FE1A   WKAREA      VS work area
FE3F   BSSTR
FE4B   SPEED       Sprite speed
FE4C   SPBASE
FE4D   MVDIST      Move.distance
FE4E   NOSPR       Number of Sprites
FE4F   DLSPNO      Number of circling Sprites
FE50   PLSPNO      Plot sprite number
FE51   MVNO
FE52   DELSPR
FE53   VCOUNT      Counter for cursor flash
FE54   VDPSTS      Copy of VDP Status register
FE55   SPRTBL      Control Buffers for Sprites
FF55   SMBYTE      Size/Magnification
FF56   LENLO
FF57   LENHI
FF58   VINTFG      Sprite interrupt flag: If 0 implies safe to
                                          write to screen
FF59   CHPTR       Character pointer


Virtual screen controls - see below

FF5D   SCRN0       FFB7   SCRN6
FF6C   SCRN1       FFC6   SCRN7
FF7B   SCRN2       FFD5   VS TYPE TABLE - Screen subfunctions
FF8A   SCRN3       FFED   VIRTUAL SCREEN
FF99   SCRN4       FFEE   OVERLAY
FFA8   SCRN5
```

Virtual Screens - Byte format for each screen

Byte No	Contents
1	Screen Type, Auto Scroll, Cursor Flash, Page Mode
2	Current print position in virtual screen
3	2nd Byte of above
4	Absolute top left-hand corner
5	2nd Byte of above
6	Size of screen in characters
7	2nd Byte of above
8	Line width of Physical screen
9	Holds Cursor character
10	Border colour, Paper, Ink
11	Print colours: Ink, Paper; Print Attributes
12	2nd Byte of above
13	Non-print colours: Ink, Paper; Non-print Attributes
14	2nd Byte of above
15	Scroll count

INTFFF (FA98 Hex 64862 Decimal)
The computer generates interrupts every 1/64th of a second. To allow the user to use these interrupts there is an interrupt flag (INTFFF) and a USERINT location. The interrupt flag determines which of the available routines are called at each interrupt.
0 = OFF, 1 = ON.

Bit 0 - Sound
    1 - Break key
    2 - Keyboard auto repeat
    3 - Sprite movement and cursor flash
    4 - USER
    5 - USER
    6 - USER

If any of the user bits are set a call is made to the USERINT location.

PRORPL If PRORPL = 1, output is sent to the device specified by IOPL
       If PRORPL = 0, output is sent to the device specified by IOPR

       IOPR 0 = Screen
       IOPL 1 = Centronics
             2 = RS232 A

KBDFLG (FA91 Hex 64145 Decimal)

Bit 7 - Alpha lock
    5 - Page/Scroll
    2 - Numeric keypad lock

CTYSLT (FB45 Hex 64325 Decimal)
This location selects a keyboard configuration and is initialised by the switches at the rear of the PCB.

APPENDIX 6

FUNCTION KEYS

The Function Keypad can be used to customise the computer for a particular application. There are eight keys marked F1 to F8.

Try this program:

```
10 PRINT ASC(INKEY$)
20 GOTO 10
```

If you press any key, you will see its ASCII code displayed and the shifted value if the shift key is pressed simultaneously.

F1	128	SHIFT	and	F1	136
F2	129	"	"	F2	137
F3	130	"	"	F3	138
F4	131	"	"	F4	139
F5	132	"	"	F5	140
F6	133	"	"	F6	141
F7	134	"	"	F7	142
F8	135	"	"	F8	143

If required, character patterns can be assigned to the function keys using the GENPAT statement.

For example,

```
10 GENPAT 1,129,32,80,136,136,248,136,136,0
```

will make F2 produce a character 'A'.

APPENDIX 7

COLOUR TABLE

0	Transparent
1	Black
2	Medium Green
3	Light Green
4	Dark Blue
5	Light Blue
6	Dark Red
7	Cyan
8	Medium Red
9	Light Red
10	Dark Yellow
11	Light Yellow
12	Dark Green
13	Magenta
14	Grey
15	White

APPENDIX 8

SOUND TABLE 1

FREQUENCY = 4000000/32*n

(Where n is the value)

DIRECT COMMAND	SBUF	RESULT (Hz)
10	80	12500
20	160	6250
30	240	4166
40	320	3125
50	400	2500
60	480	2083
70	560	1785
80	640	1562
90	720	1388
100	800	1250
110	880	1136
120	960	1041
130	1040	961
140	1120	892
150	1200	833
160	1280	781
170	1360	735
180	1440	694
190	1520	657
200	1600	625
210	1680	595
220	1760	568
230	1840	543
240	1920	520
250	2000	500
260	2080	480
270	2160	462
280	2240	446
290	2320	431
300	2400	416
310	2480	403
320	2560	390
330	2640	378
340	2720	367
350	2800	357
360	2880	347
370	2960	337
380	3040	328
390	3120	320

DIRECT COMMAND	SBUF	RESULT (Hz)
400	3200	312
420	3360	297
440	3520	284
460	3680	271
480	3840	260
500	4000	250
520	4160	240
540	4320	231
560	4480	223
580	4640	215
600	4800	208
620	4960	201
640	5120	195
660	5280	189
680	5440	183
700	5600	178
720	5760	173
740	5920	168
760	6080	164
780	6240	160
800	6400	156
820	6560	152
840	6720	148
860	6880	145
880	7040	142
900	7200	138
920	7360	135
940	7520	132
960	7680	130
980	7840	127
1000	8000	125
1020	8160	122

SOUND TABLE 2

NOISE

DC (periodic noise)	SB	R
0	0	Shift rate = 7812.5 Hz
1	8	Shift rate = 3906.25 Hz
2	16	Shift rate = 2604.17 Hz
3	24	Shift rate = CHANNEL 2
Pink Noise		
4	32	Shift rate = 7812.5 Hz
5	40	Shift rate = 3906.25 Hz
6	48	Shift rate = 2604.17 Hz
7	56	Shift rate = CHANNEL 2

SOUND TABLE 3

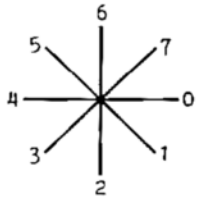
VOLUME

Direct Com.	SBUF	Result (DB)
0	0	OFF
1	16	-28
2	32	-26
3	48	-24
4	64	-22
5	80	-20
6	96	-18
7	112	-16
8	128	-14
9	144	-12
10	160	-10
11	176	- 8
12	192	- 6
13	208	- 4
14	224	- 2
15	240	- 0

APPENDIX 9

ABSOLUTE DIRECTIONS

Some graphics commands, including MVSPR, and VIEW, use a direction parameter to specify one of seven directions. These are illustrated in the diagram below.



APPENDIX 10

FLOWCHART CONVENTIONS

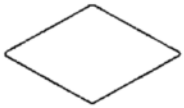
Manual Operation

Input/Output

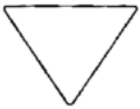
Process

Terminal/Interrupt

Preparation



Decision

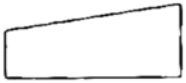


Merge

Document

Display

Connector



Manual Input

Off Page

Communications Link

GLOSSARY OF TERMS

Absolute Address	Information or data held in a computer is found by the address of its location. In machine code programs, the number defining an address is called an absolute address.
AC	Alternating Current.
Access Time	How long it takes to reference an item in memory.
Accumulator	A type of register.
ADC	Analogue to digital converter. Converts analogue signals into digital signals, would you believe! There are also digital to analogue converters, which work in the opposite direction.
Address	Each memory location has an address, used to find data or a program instruction.
Algorithm	A set of steps for performing a task.
Alphanumeric	Numbers, letters and sometimes other things.
Array	An arranged set of values linked by some kind of logical relationship. Each element in an array has a unique reference.
ASCII	American Standard Code for Information Interchange. Pronounced 'Askey', it's just a way of representing alphanumeric characters in binary. Difficult to get away from this one, it crops up all over the place.
Assembler	A programming language one step away from the zeros and ones the computer understands and uses. Assembly code is the coding for a program written in assembler.
Backup	When things go wrong, if you haven't got one, you're in trouble.

BASIC	The Beginner's All-Purpose Symbolic Instruction Code.
Baud Rate	Number of bits per second transmitted along a line.
BCD	Binary Coded Decimal. A way of expressing decimal numbers using bits. Uses four binary bits for each decimal number.
Benchmark	A standard set of tests for seeing how fast a computer can perform. Used mainly in comparing one computer with its rivals.
Binary	Number system using only two digits, 1 and 0.
BIT	Binary digit. Either a zero or a one, it is the basic unit of information storage.
Boolean Algebra	Set of logical instructions written using algebra, with an answer either TRUE or FALSE.
Bootstrap	A set of instructions held permanently in the computer which have to be loaded before the computer can load programs.
Branch	In programming terms, a branch is a part of a program where a decision is made and the program flow is transferred depending on the result. This is a conditional branch. An unconditional branch is something like the GOTO statement, where the program control jumps somewhere else without a decision being made.
Buffer	Somewhere data is stored temporarily, until the CPU is ready to process it. Also used to allow one part of the computer to work at a different speed from another part.
Bug	We all get these, so don't worry. A software error.
Bus	A set of connections which allow a route around the computer for signals.
Byte	A set of bits, the smallest unit that means anything. One byte is normally represented by 8 bits, and represents a character or number.

Centronics	A manufacturer of printers. Very popular. Lucky you've got a centronics type interface.
Character Set	The set of characters (sorry)!
Chip	This is what most people call an integrated circuit. It's a tiny piece of silicon, and the bread and butter of computers. (No jokes please.)
Command	An instruction to the computer to tell it to do something.
Compiler	Translates source code into object code.
Constant	Something (either a number, or a string) which doesn't change.
CP/M	Stands for Control Program/Monitor. A widely used and well recognised operating system which makes available to you a wealth of software packages. If you take computing seriously it's well worth the investment, there are books around which describe it fully, if you're interested.
CPU	The Central Processing Unit is a complex chip where all the logical and arithmetic operations are carried out. It's your computers 'brain'.
Crash	Something that happens to programs. When a program crashes it's because the computer has encountered an instruction which has totally confused it, so instead of getting an error message you usually get nothing, or lots of rubbish displayed on the screen.
Cursor	The cursor tells you where the character you are about to type will appear. It's the blob on the screen that's about the size of an ordinary character.
Data	Data is information which can be processed, stored or produced by a computer.
DC	Stands for Direct Current. A constant voltage.
Debug	The identification and removal of errors from a program.

Disc	An L.P. shaped plate covered in magnetic material which can store information or data on its concentric tracks. Discs have a fast access time, because the read/write head can position itself quickly over the required data without having to read all the preceding storage area.
Dump	To make a backup of a section of memory by printing it, or sending it to a backing store, to give a security copy usually.
Edit	To change data from what it was to what you want it to be.
Emulator	Software which enables one computer to duplicate the instruction set of another.
EOF	Stands for End Of File.
EPROM	Erasable, Programmable Read-Only Memory.
Error Message	We all see lots of these, a code or message to tell you that you have made a mistake.
Execute	The carrying-out of a program or single instruction.
Execution Time	How long it takes.
File	A file is a block of data organised so that it can be stored and retrieved as required. Files always have names.
Flag	An indicator used to indicate something about data. For instance the Z80 CPU has a flag which tells you whether the last operation performed resulted in zero or non zero.
Floppy Disc	Cheap, flexible store for data.
Flowchart	A graphic way of representing the order of a set of events.
Gate	A single logic function.
Gigo	Garbage in, garbage out! Antiquated expression, but I like it.

Glitch	A spike of electrical noise. You don't want any of these. Can destroy your memory contents.
Hard Copy	A paper printout of your program or data is called hard copy.
Hardware	Hardware is the physical bits and pieces (chips etc.) that make up your computer.
Hertz (Hz.)	Measure of frequency meaning cycles per second.
Hex	In everyday mathematics we use decimal, or base 10. Hexadecimal is a number system in base 16 and uses the numbers 0 to 9 and letters A to F (representing 10 through 15).
Input	Information placed into the computer's memory is input data, and may originate from, for example, the keyboard.
I/O	Abbreviation of Input/Output.
Integer	A whole number.
Interface	Software or hardware, or both, used to enable the computer and a peripheral to talk to each other.
Joystick	Used mainly to enable games to be played on a computer. We all know what a joystick is anyway, don't we?
Kilo (K)	Generally means one thousand, except when referring to memory size when it means 1024.
Line Number	The number required at the beginning of a line in BASIC is its line number. The program is always executed in line number order, unless you use something like a GOTO or GOSUB statement.
LOAD	The placing of data in memory from a backing store or program.
Location	Same as absolute address.
Machine Code	Literally the language the computer understands. Machine code is the language all other languages have to be translated into before the computer can execute a program.

Memory	Storage inside the computer for data and programs, measured in bytes.
Menu	List of choices open to the user, usually encountered as the first page, or screen of a program.
Microcomputer	A small computer using a microprocessor chip. In the MTX series computers, the microprocessor is the Zilog Z80.
Microprocessor	The chip used in your computer as its CPU. Microprocessors crop up everywhere these days, in ovens, Hi Fi equipment, they are even responsible for telling you to put your seatbelt on in a Maestro.
Microsecond (us)	One millionth of a second.
Millisecond (ms)	One thousandth of a second.
Monitor	Think of it as a high definition television that can be used only as a display screen.
Nanosecond (ns)	One billionth of a second. (One billion is 1,000,000,000 or 10^9.)
Nibble	Half a byte, i.e. usually four bits.
Non-Volatile	Most of the contents of memory are lost when the power is turned off. Non-volatile memory doesn't disappear. For example, the information in ROM is non-volatile.
Null String	An empty string. The string must exist, and it must have nothing in it for it to be a null string.
Number-Crunching	Performing complex calculations quickly.
Object Code	A form of code the computer understands. If you write your program in a high level language, (source code) it has to be translated into object or machine code before the computer can act on it. This is a binary version of the source code and is produced by the compiler.
On-Line	Peripherals connected to and communicating with a computer are said to be on-line.

Operand & Operator	Machine code instructions can be divided into these two parts. The operator is the process which is carried out, e.g. add, subtract, etc. and the operand is the data the process is carried out on, usually a number.
Operating System (OS)	Software which supervises the running of other programs. CP/M, developed by Digital Research Inc. in 1976 is an excellent operating system for use with Z80 microprocessor computers like the MTX series.
Output	The results that the computer makes available to the user (either on the screen or as a printout, maybe).
Overflow	When the space allowed for the answer of an arithmetic expression is too small, an overflow condition will occur. The Z80 CPU has an overflow flag.
Pack	A way of compacting information to economise on storage space inside a computer.
Page	A block of data, as displayed by the television set or monitor. Sometimes a page is made up with several frames, or screens, of data.
Paging	Switching between blocks of computer memory.
Peek	A BASIC command which allows you to read the contents of a specified memory address.
Peripherals	Devices linked to the computer to enable it to gather and display information; e.g. a printer, or a TV screen are peripherals.
Pixel	Picture element. It's the smallest area of display that the computer can control. The more pixels you've got, the higher the resolution of your computer.
Poke	BASIC command which places integer values into a specified memory location.
Port	A socket on the computer into which an I/O device can be plugged.

Program	A set of instructions which the computer carries out.
PROM	Programmable Read Only Memory.
RAM	Random Access Memory.
Record	A grouped set of related data or information. A file is generally made up of lots of records.
Register	A special storage location in the CPU which holds data on which calculations are performed.
Reset	On the MTX series computers the two keys on either side of the space bar are the Reset Keys. Reset means the same as Initialise, and once pressed, the computer returns to the state it was in when you first switched on.
Reserved Word	A word that has a specific meaning to the compiler, so it cannot be used as a variable name in a program.
ROM	Permanent Read Only Memory.
RS232	A type of interface.
Run	A command used to tell the computer to execute a program.
Scroll	The continuous movement of the display on the screen. Usually scrolling means that the latest line entered is added at the bottom and all the other lines move up one, causing the top line to dissappear from view.
Software	The program itself, i.e. as opposed to Hardware.
Source Code	What is actually written by the programmer before it is converted to object or machine code.
String	A sequence of records; words, letters or numbers.

Subroutine	Often a part of a program will need to be repeated several times during the 'run'. Instead of writing the section each time you need it, a subroutine means you can write it just once, and 'call' or use it as needed.
Syntax	Computer languages are very precise. Statements need to be in the correct order in the program, or it will crash. The rules which decide the grammar of the language are its Syntax.
Variable	An element of a program that can have various values. It is a label used to refer to an area of memory.
Volatile	Opposite of non-volatile.
Zilog	"The last word in integrated logic". The manufacturer of the Z80 micro chip used in the MTX series computers.

MTX SERIES TECHNICAL APPENDICES

1 Introduction
  Overall Description
2 Technical Specification
3 System Bus
4 System Block Diagram
5 Electronic Circuit Schematic
6 Video Display Processor
7 Sound Generator
8 Memory Maps
9 Input/Output Port Summary
10 Parallel Printer Interface
11 Parallel Input/Output Port
12 Memotech DMX80 Printer Connector

1 INTRODUCTION

Overall Description

The MTX500 Series personal computer systems are high performance 8-bit computers uniquely designed to operate in memory intensive ROM-based or DISC-based environments. The choice of the Z80A Microprocessor and the TMS 9918A series video processor as the key components of the hardware architecture is consistent with a low cost ROM-based system with colour TV output plus the capability to expand to accomodate a fully RAM-based Disc operating system such as CP/M, utilising a high quality 80 column colour monitor output.

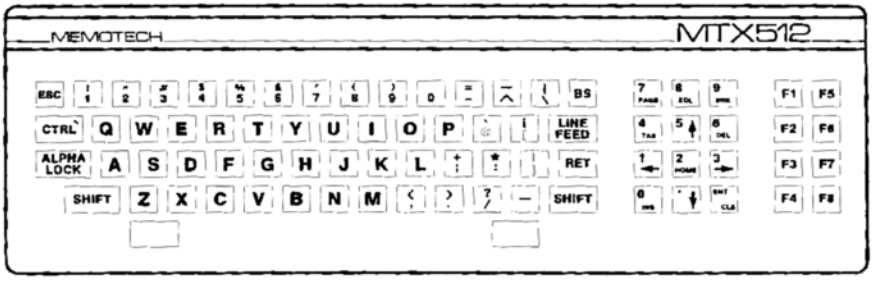
The memory size can be either 32K or 64K Bytes as standard, expandable to 512K Bytes. There is a separate 16K Byte dedicated video memory. A 24K Byte ROM contains MTX - BASIC, the system monitor, supplementary languages and utilities. The standard interfaces included are tape cassette (Read/Write to 2400 baud), Keyboard, Cartridge Port, Twin Joysticks, Parallel Centronics type printer port, uncommitted Parallel Input/Output port, Colour TV output with sound, composite video output - monochrome or colour, and audio output. Optional interfaces include a completely independent twin RS232C with buffered bus extension, Colour 80 Column Board, Floppy Disc System, Silicon disc fast access RAM boards, and a Winchester Disc System.
The Keyboard consists of 79 full travel typewriter style keys mounted on a steel base plate which is fitted to the Aluminium enclosure. Aluminium was chosen for good heat dissipation, durability and RFI shielding.

2 TECHNICAL SPECIFICATION

Hardware

Chassis
Two front-hinged black anodised brushed aluminium extrusions are separated at the rear by a black plastic moulding. The extrusions act as heatsinks for the voltage regulation circuitry. Two matt black powder coated stamped aluminium endplates, are secured by 3 screws each.
Dimensions in millimetres: Width 488 Depth 202 Height 56
Weight: 2.6 kilograms

Keyboard
A 1mm mild steel sheet is bolted to the upper chassis and supports 79 keys which are interconnected by an independent p.c.b. The keys are arranged as:
Standard U.K. QWERTY layout with 57 professional typewriter keys, standard pitch and spacing. Keys F and J are recessed for easy fingertip location wherever possible. Foreign language keyboards are available.
Twelve dual function keys are arranged as a separate numeric keypad with cursor control and editing keys.
Eight function keys (programmable in conjunction with shift to provide 16 user definable functions).
Two unmarked keys, which must be depressed simultaneously to reset the computer.
Auto repeat is standard on the alpha-numeric keys.



CPU Board
Mounted in the lower chassis, the CPU board accommodates:
Zilog Z80A CPU operating at 4MHz.
24K of ROM which contains:
MTX BASIC - incorporating sophisticated MTX LOGO-type graphics commands.
MTX NODDY - interactive screen manipulation routines.
FRONT PANEL DISPLAY - incorporating Z80 Assembler/Disassembler plus Z80 Register, Memory and Program display and manipulation routines.
VIDEO DISPLAY PROCESSOR - with 16K dedicated video-RAM.
USER-RAM - 32K on the MTX500 and 64K on the MTX512. User RAM size is constant under all display formats.
VIDEO BOARD - for television and sound signal encoding.
REAL TIME CLOCK
CHARACTER SETS - Numeric, upper case, lower case, user-definable characters and user-definable sprites. Resident international character sets and appropriate keyboard layouts for UK, USA, France, Germany, Spain and Sweden. Character sets for Denmark and Italy are also available.

Expansions
Up to two expansion boards may be added internally. These may be Memory (RAM) Boards or the Communications Board.
MEMORY BOARDS
RAM may be increased by the addition of boards which provide 32K, 64K, 128K or 256K of memory, up to a maximum of 512K.
COMMUNICATIONS BOARD
Available as an internal expansion, this board carries two completely independent RS232 interfaces (running at up to 19 200 baud) with full handshaking and modem communication lines, and also the disc drive bus. The Communications Board is required to run the FDX and HDX disc based systems and the MTX Node/Ring System.
NODE/RING SYSTEM - Communications software and interfacing enabling construction of MTX Ring Systems. The system is interrupt driven and runs in conjunction with the twin RS232 Communications Board.
Compatibility of the memory boards and Communications Board is given below.

Compatibility table of internal expansion boards

	RAM boards				
	32K	64K	128K	256K	Comms. board
32K		*	*	*	*
64K	*	*	*	*	*
128K	*	*	*	*	*
256K	*	*	*	*	*
Comms. Board	*	*	*	*	

*=compatible

ROM Expansions
Via the cartridge port or disc drive bus these provide:
MTX PASCAL
MTX FORTH
NODE SYSTEM software
Business, Education and Games software

Display
Colour TV and/or Video Monitor
40 column x 24 line display as standard, with optional Colour 80 column board. (FDX or HDX disc based system required).
Display Facilities:
FULL SCREEN HANDLING
EIGHT USER DEFINABLE VIRTUAL SCREENS
SCREEN FORMATS
Text: 40 x 24 characters
Text with graphics: 32 x 24 text with 256 x 192 pixels in 16 colours

Graphics Facilities
Up to 32 independently controllable user definable sprites, plus pattern plane and backdrop plane. High level sprite-orientated graphics commands.

Input/Output
Provided as standard:
CASSETTE PORT (variable rate, up to 2 400 baud)
UNCOMMITTED PARALLEL INPUT/OUTPUT PORT
TWO JOYSTICK PORTS with industry standard pin-outs
FOUR CHANNEL SOUND UNDER SOFTWARE CONTROL - three independent voices plus pink noise output through TV speaker, or through separate Hi-Fi output
MONITOR OUTPUT - composite video signal (1V peak to peak)
CARTRIAGE PORT
PARALLEL PRINTER PORT (compatible with Centronics-type printers)
Available as an expansion:
COMMUNICATIONS BOARD WITH TWO RS232 INTERFACES and disc drive bus
Suitable Printers
Centronics-type parallel printers
RS232 serial printers (requires Communications Board)

Power Supply Unit
Input: 220/240 VAC 50/60 Hz. or 110/115 VAC 50/60 Hz.
Output: 22.5 VAC, 1A tapped at 18V and 9V.
Dimensions in millimetres: Width 92 Depth 110 Height 70
Weight: 1.0 kilogram
The PSU is double insulated and has a side mounted rocker switch which is internally illuminated when the unit is on. The mains transformer is located between two groups of four anti-vibration, noise absorbing rubber mounts. Extensive strain relief mouldings are incorporated in the PSU casing to support the input and output cables. The output cable terminates in a 240 degree, six pin DIN connector. The PSU is supplied as a sealed unit.

MTX Series Disc Based Systems
These are the:

FDX Floppy Disc System
and the
HDX Winchester Disc System

Both of these systems require the Communications Board expansion within the MTX computer, and a minimum of 64K RAM. Both systems have the following features:
A 19 inch wide chassis comprising four black anodised brushed aluminium extrusions. Black powder coated end plates are each secured by six screws. The chassis contains a card cage which can accommodate:
One computer expansion board
One Colour 80 column board
Four silicon disc memory boards
One floppy disc controller board
An integral power supply which also powers the MTX computer.
Inputs can be 240/220 VAC 50/60 Hz or 110/115 VAC 50/60 Hz.
Parallel port for bus expansion
Two slots are provided on the front face for horizontally mounted five and a quarter inch disc drives.
External battery backup facilities are optionally available
A license to use the Digital Research Inc. CP/M 2.2 operating system is provided with the FDX and HDX systems, as is CP/M itself.

Colour 80 column board
Mounted in the FDX or HDX systems the board permits the use of colour programs requiring an 80 column screen running under CP/M 2.2, such as Colour Wordstar. Also available is the wide range of existing CP/M based software.

80 Column board-Input and Output
RGB monitor output with selectable positive/negative sync.
Monochrome composite video output, 1V peak to peak, negative sync.
Light pen input
Single channel sound
Screen display formats:
80 columns x 24 lines text (max)
160 x 96 graphics mode
Two alternate 96 element character sets with true lower case descenders.
4K ROM based graphics characters
Teletext compatibility
High speed glitch free screen update (average 25 000 baud)
The Colour 80 column board provides a complete emulation of a CP/M terminal via ROM software, and features:
Full cursor control
Vector plot, point plot
Powerful editing facilities with screen dump
Complete attribute control for colour and monochrome displays

Silicon Discs

These are a quarter or one megabyte fast access RAM boards which are full emulators of CP/M drives 0 to 13. Four silicon discs may be mounted within the HDX or FDX chassis, providing from one to four megabytes per card frame. However, the silicon disc controllers can supervise four logical drives, of up to eight megabytes each giving a maximum silicon storage of 32 megabytes. This is in addition to the four five and a quarter and/or eight inch conventional floppy disc drives handled by the floppy disc controller board. Numerous advantages include:

Speed - up to five times faster than a Winchester disc, and fifty times faster than a floppy disc.

A dramatic increase in efficiency of proven eight bit CP/M software to 16/32 bit software levels, obviating the need for complex and costly memory management techniques

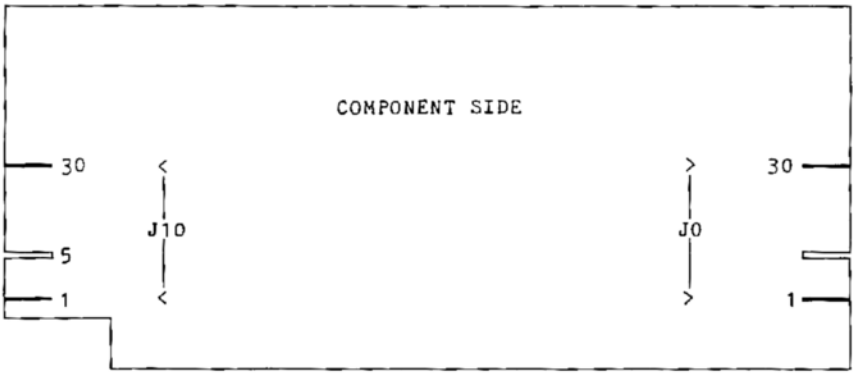
Permits single floppy disc CP/M system which is ideal for database manipulation, word processing and compilation.

Greatly reduces disc wear and prolongs life of mechanical disc drives, enhancing reliability especially in disc intensive transactions.

Floppy Disc Controller Board

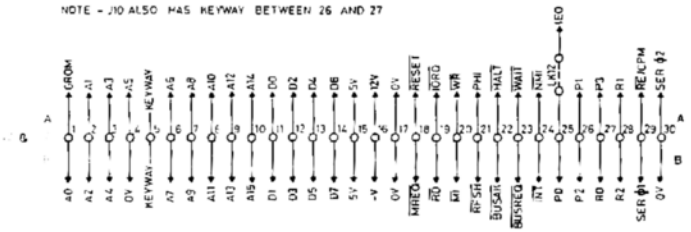
This board uses the full Western Digital 1791 chip set and supports most CP/M floppy drives, types 0 to 13, which range from single sided single density five and a quarter inch floppies to double sided double density eight inch floppies, using SASI (Shugart) standard interfaces. Any combination of four SASI compatible drives can be controlled. The WD 1791 controller set together with a bipolar DMA controller provides a high speed processor interface minimising latency and facilitating rapid data transfer especially on high capacity discs. Variable and fixed write precompensation is software selectable. Bus extenders permit the connection of external floppy drives.

3 MTX SERIES SYSTEM BUS



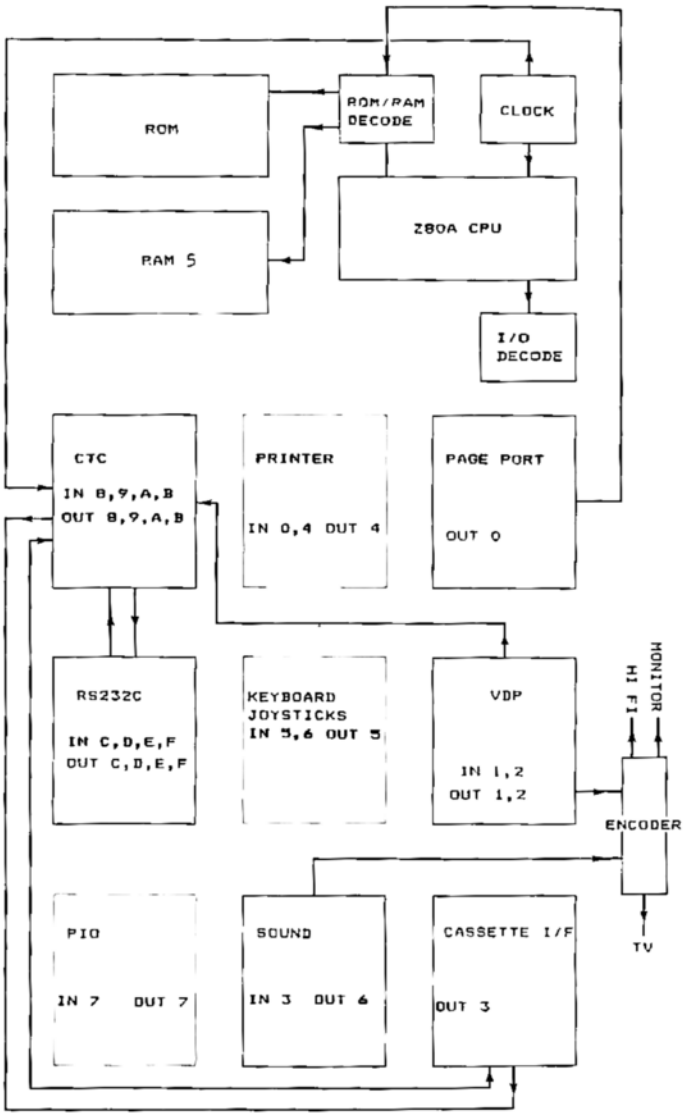
The system Bus comprises the full Z80A bus, power supply rails, ROMpak enable (GROM), ROM page ports R0 to R2, RAM page ports P0 to P3 and serial clock lines 01 and 02.

All lines are externally available on J10, which is a 60 way (30 + 30) 0.1" card edge plug, or internally on J0 which is also a 0.1" 60 way card edge plug.

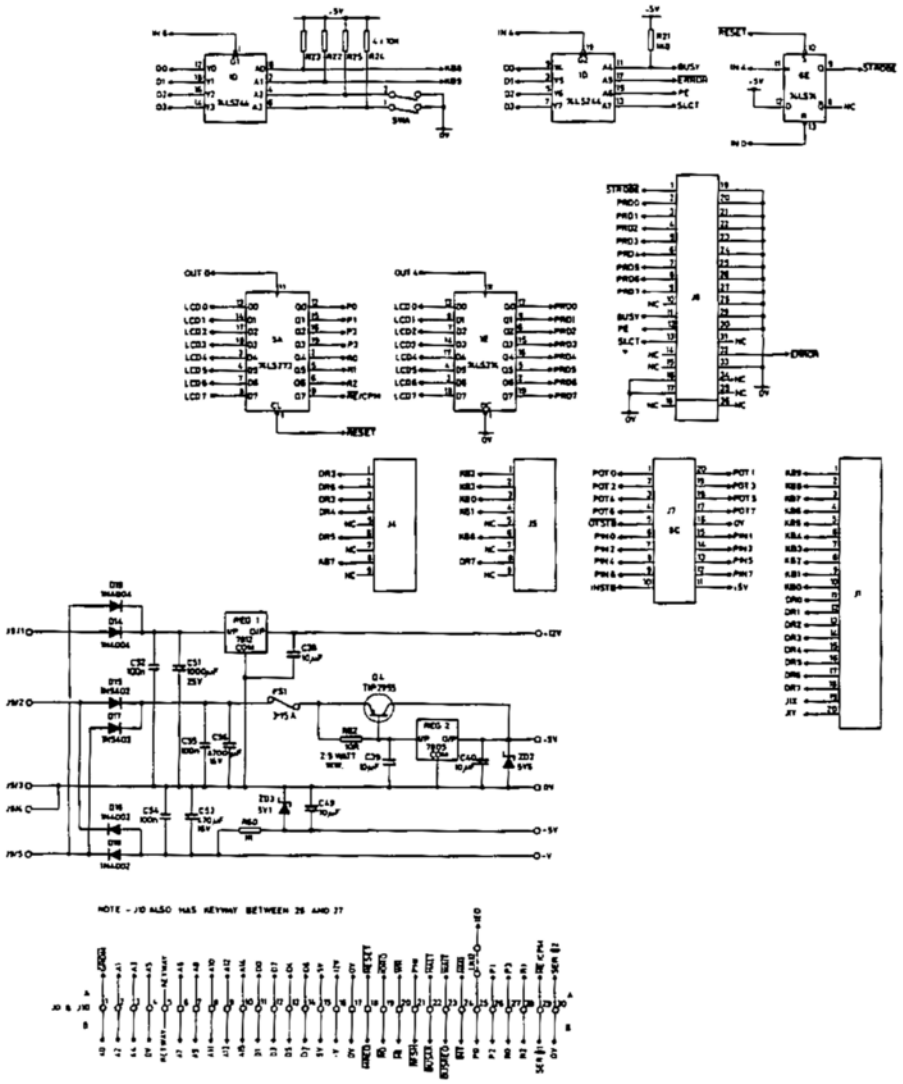


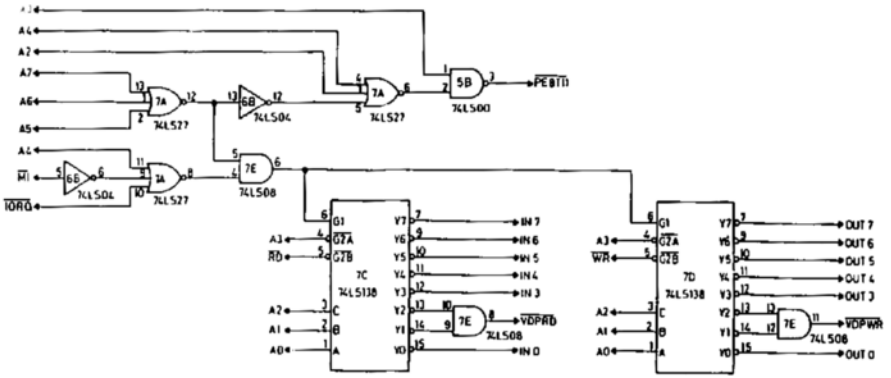
Note: (1) J10 is a mirror image of J0
(2) Component side = A
Solder side = B

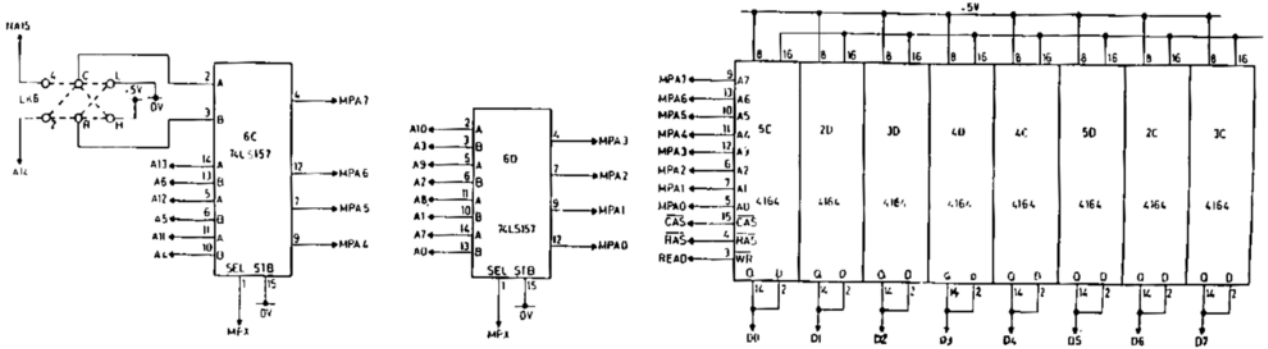
4 SYSTEM BLOCK DIAGRAM

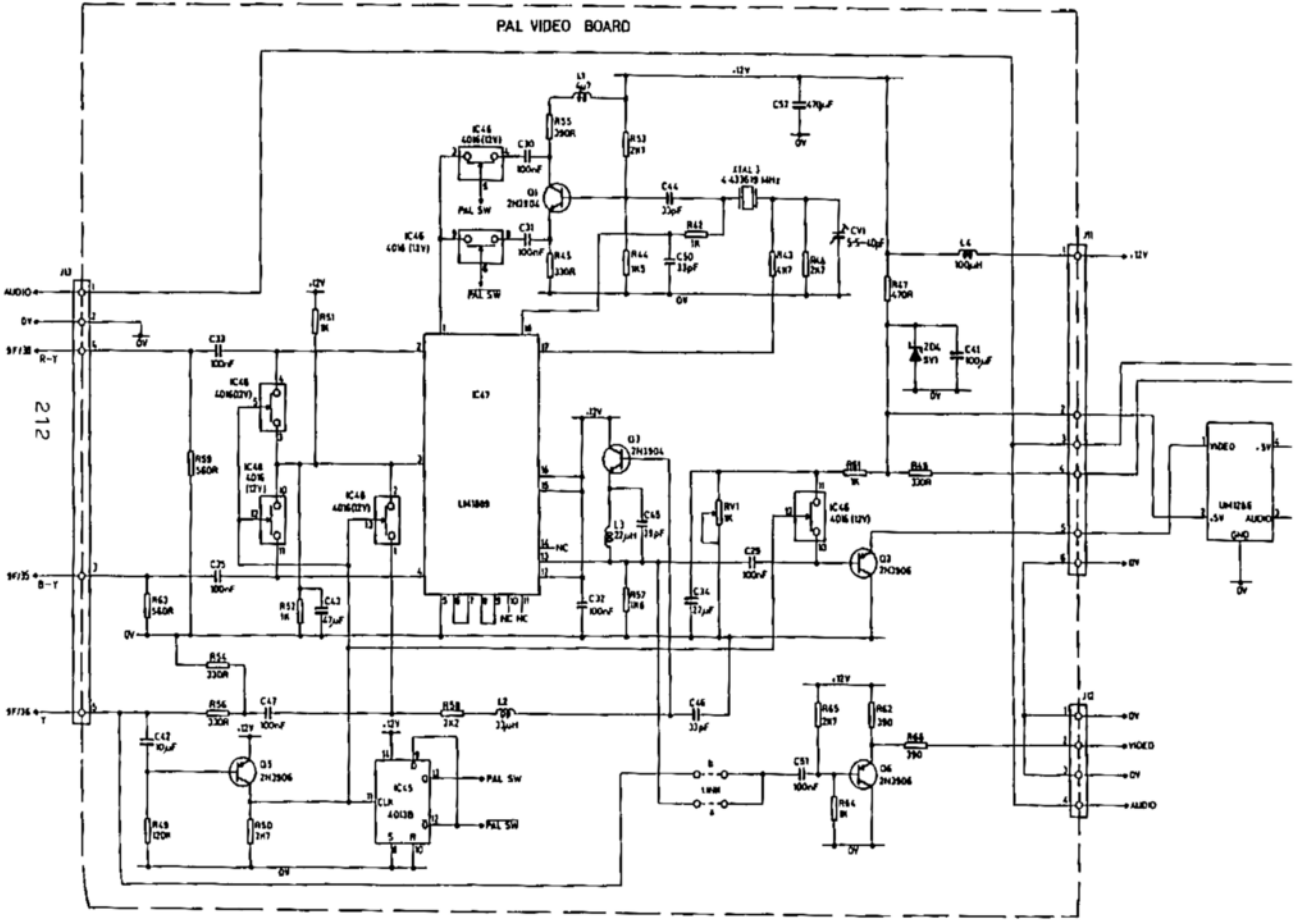


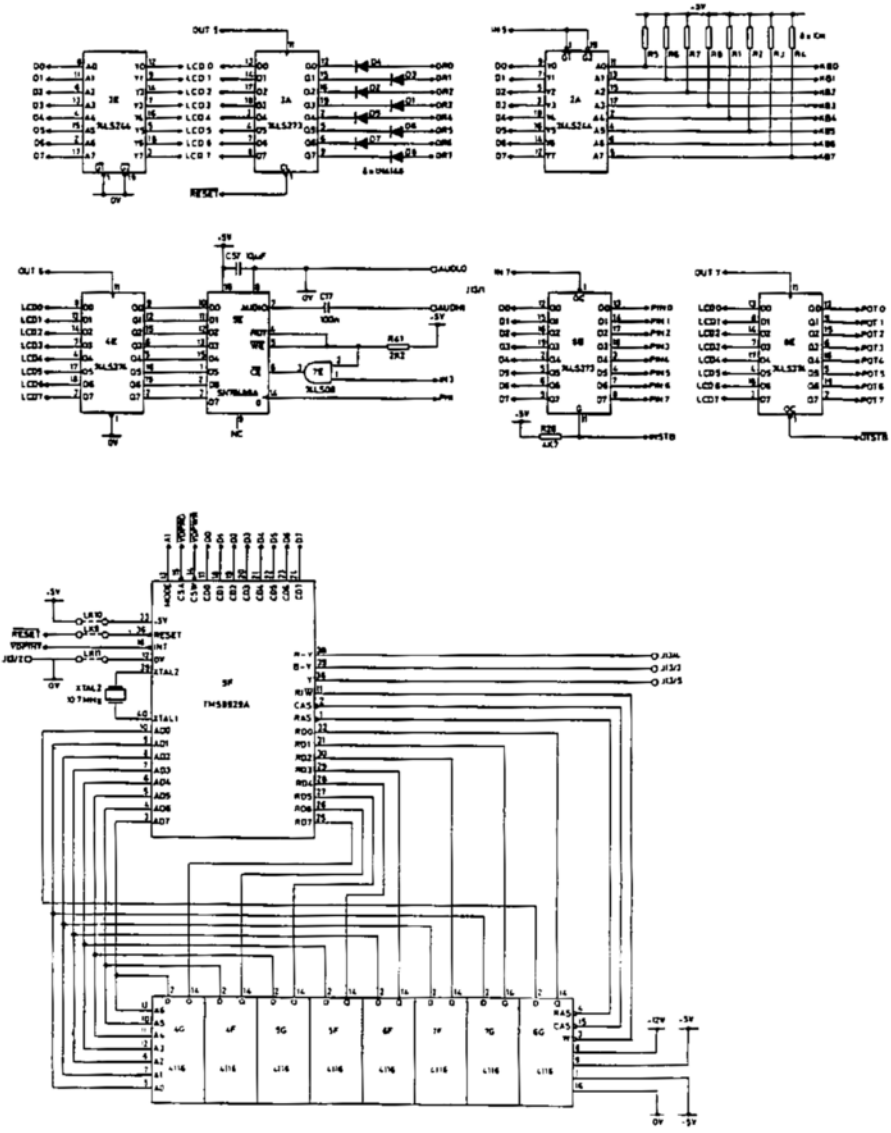
5 ELECTRONIC CIRCUIT SCHEMATICS

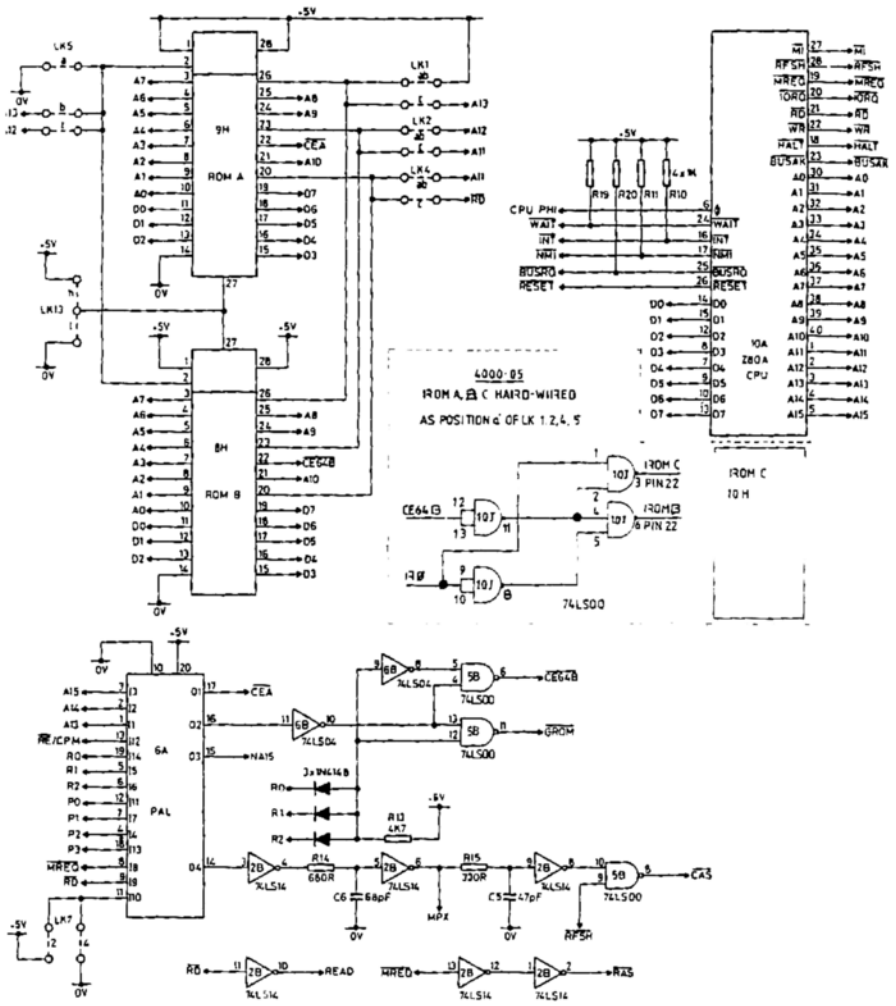


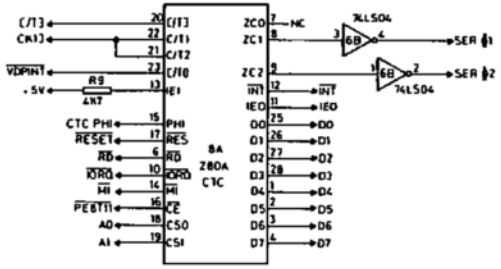

A4
A2
A7
A6
A5
A3
M1
IORQ
7A
74LS27
6B
74LS04
5B
74LS00
7E
74LS08
7C
74LS138
7D
G1
G2A
G2B
A
B
C
Y0
Y1
Y2
Y3
Y4
Y5
Y6
Y7
RD
WR
IN 7
IN 6
IN 5
IN 4
IN 3
IN 0
OUT 7
OUT 6
OUT 5
OUT 4
OUT 3
OUT 0
VDPRD
VDPWR

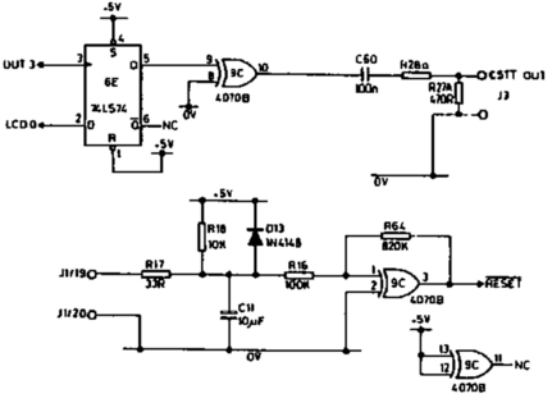

MA15
MA14
LKB
-5V
0V
6C
74LS157
SEL
STB
MPA7
MPA6
MPA5
MPA4
MPA3
MPA2
MPA1
MPA0
6D
5C
2D
3D
4D
4C
5D
2C
3C
4164
CAS
RAS
WR
READ
D0
D1
D2
D3
D4
D5
D6
D7

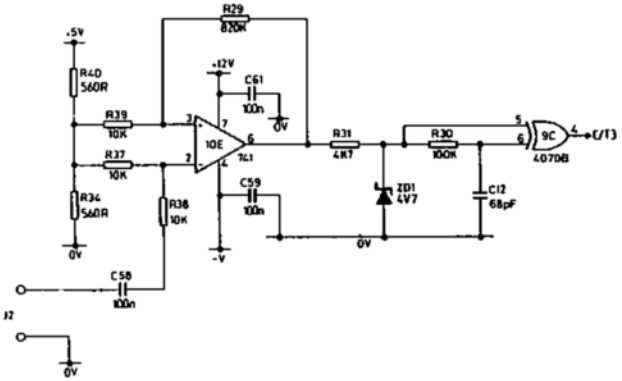

PAL VIDEO BOARD
J3
AUDIO
0V
9V/38
R-Y
9V/35
B-Y
9V/36
Y
+12V
IC47
LM1889
IC46
4016(12V)
IC48
4016(12V)
IC45
4013B
CLK
PAL SW
XTAL 1
4.433619 MHz
Q1
2N3904
Q2
2N3906
Q3
2N3906
Q4
2N3906
Q5
2N3906
Q6
2N3906
Q7
2N3904
ZD1
5V1
CV1
5-40pF
J1
J2
+12V
VIDEO
0V
AUDIO
LM1286
.5V
GND
212

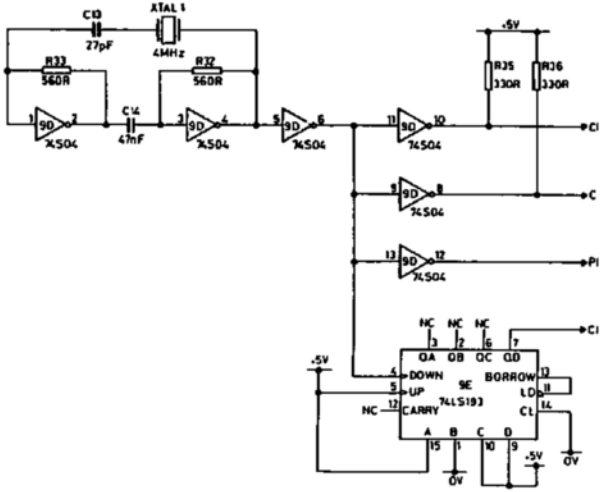



ROM A
ROM B
9H
8H
ROM C
10H
Z80A CPU
PAL
74LS00
74LS04
74LS14
4000:05
ROM A, B & C HARD-WIRED
AS POSITIONS OF LK 1,2,4, 5


C/T3
CLK
VDPINT
+5V
R9
4K7
CTC PHI
RESET
RD
IORQ
MI
PESTI1
A0
A1
C/T3
C/T1
C/T2
C/T0
IEI
PHI
RES
RD
IORQ
MI
CE
CS0
CS1
8A
Z80A
CTC
ZC0
ZC1
ZC2
INT
IEO
D0
D1
D2
D3
D4
D5
D6
D7
NC
74LS04
6B
SER I1
SER I2

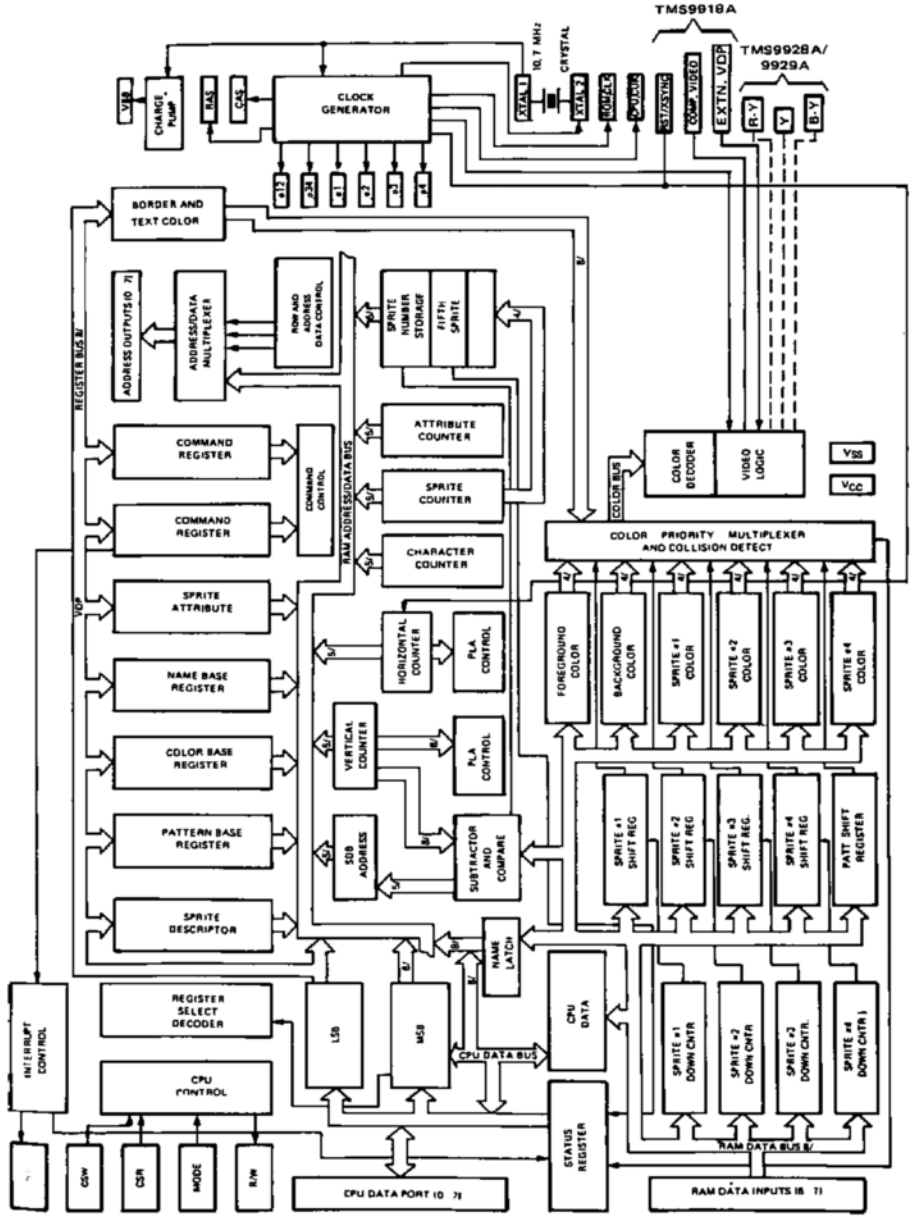

+5V
OUT 3
LCD0
6E
74LS74
NC
0V
9C
4070B
C60
100n
H28a
CSTT OUT
R77A
470R
J3
R18
10K
D13
W4148
R64
820K
J1r19
R17
33R
R16
100K
RESET
J1r20
C11
10µF
NC


R29
820K
+5V
R40
560R
+12V
C61
100n
R39
10K
R37
10K
R34
560R
R38
10K
10E
741
R31
4K7
R30
100K
9C
4070B
C/T3
C59
100n
ZD1
4V7
C12
68pF
C58
100n
J2
-V
0V


C13
27pF
XTAL 1
4MHz
R33
560R
R32
560R
C14
47nF
9D
74S04
R35
330R
R36
330R
CI
C
PI
NC
QA
QB
QC
QD
DOWN
UP
CARRY
BORROW
LD
CT
74LS193
A
B
C
D
+5V
0V

6 VIDEO DISPLAY PROCESSOR TMS9918 SERIES

The Video Display Processor (VDP) used in the MTX Series is the TMS9918 Series. The TMS9929A is used in computers for the European market, and the TMS9928A is used for North America. The VDP is I/O mapped at ports 1 and 2. (MODE = 0 for port 1, and MODE = 1 for port 2.) The colour difference signals are encoded, mixed with sound and fed to the appropriate RF modulator, dependent upon the country for which the machine is intended.


TMS9918A
TMS9928A/
9929A
10.7 MHz
CRYSTAL
CLOCK GENERATOR
CHARGE PUMP
VBB
RAS
CAS
XTAL 1
XTAL 2
MODE
CPUCLK
INT/SYNC
COMP VIDEO
EXTN. VDP
R-Y
Y
B-Y
BORDER AND TEXT COLOR
ADDRESS OUTPUTS (0-7)
ADDRESS/DATA MULTIPLEXER
ROW AND ADDRESS DATA CONTROL
SPRITE NUMBER STORAGE
FIFTH SPRITE
REGISTER BUS
COMMAND REGISTER
COMMAND REGISTER
SPRITE ATTRIBUTE
NAME BASE REGISTER
COLOR BASE REGISTER
PATTERN BASE REGISTER
SPRITE DESCRIPTOR
COMMAND CONTROL
RAM ADDRESS/DATA BUS
ATTRIBUTE COUNTER
SPRITE COUNTER
CHARACTER COUNTER
HORIZONTAL COUNTER
PLA CONTROL
VERTICAL COUNTER
PLA CONTROL
SUB ADDRESS
SUBTRACTOR AND COMPARE
NAME LATCH
COLOR BUS
COLOR DECODER
VIDEO LOGIC
VSS
VCC
COLOR PRIORITY MULTIPLEXER AND COLLISION DETECT
FOREGROUND COLOR
BACKGROUND COLOR
SPRITE #1 COLOR
SPRITE #2 COLOR
SPRITE #3 COLOR
SPRITE #4 COLOR
SPRITE #1 SHIFT REG
SPRITE #2 SHIFT REG
SPRITE #3 SHIFT REG.
SPRITE #4 SHIFT REG
PATT SHIFT REGISTER
INTERRUPT CONTROL
REGISTER SELECT DECODER
CPU CONTROL
LSB
MSB
CPU DATA BUS
CPU DATA
SPRITE #1 DOWN CNTR
SPRITE #2 DOWN CNTR
SPRITE #3 DOWN CNTR.
SPRITE #4 DOWN CNTR
STATUS REGISTER
RAM DATA BUS
CSW
CSR
MODE
R/W
CPU DATA PORT (0-7)
RAM DATA INPUTS (0-7)

CPU Interface Control Signals
The type and direction of data transfers are controlled by the CSW, CSR and MODE inputs. CSW is the CPU-to-VDP write select. When it is active (low), the 8 bits on D7-D0 are strobed into the VDP. CSR is the CPU-from-VDP read select. When it is active (low),the VDP outputs 8 bits on D7-D0 to the CPU. CSW and CSR should never be simultaneously low. If both are low, the VDP outputs data on D7-D0 and latches in invalid data.

MODE determines the source or destination of a read or write data transfer. MODE is normally tied to a CPU low order address line.

CPU WRITE TO VDP REGISTER
The VDP has eight write-only registers and one read-only status register. The write-only registers control the VDP operation and determine the way in which VRAM is allocated. The status register contains interrupt, sprite coincidence and fifth sprite status flags.

Each of the eight VDP write-only registers can be loaded using two 8-bit data transfers from the CPU. Table 1 describes the required format for the two bytes. The first byte transferred is the data byte, and the second byte transferred controls the destination. The most-significant bit of the second byte must be a '1'. The next four bits are '0's, and the lowest three bits make up the destination register number. The MODE input is high for both byte transfers.

To rewrite the data for an internal register after a byte of data has been loaded, the status register must be read so that internal logic will accept the next byte as data and not as a register destination. This situation may be encountered in interrupt-driven program environments. Whenever the status of VDP write parameters is in question, this procedure should be used. Note that the CPU address is destroyed by writing to the VDP register.

CPU WRITE TO VRAM
The CPU transfers data to the VRAM through the VDP using a 14-bit autoincrementing address register. Two-byte transfers are required to set up the address register. A one-byte transfer is then required to write the data to the addressed VRAM byte. The address register is then autoincremented. Sequential VRAM writes require only one byte transfer since the address register is already set up. During setup of the address register, the two most significant bits of the second address byte must be '0' and '1' respectively. MODE is high for both address transfers and low for the data transfer. CSW is used in all transfers to strobe the 8 bits into the VDP. See Table 1.

TABLE 1 - CPU/VDP DATA TRANSFERS

OPERATION	MSB 7	6	5	BIT 4	3	2	1	LSB 0	CSW	CSR	MOD
WRITE TO VDP REGISTER											
Byte 1 Data Write	D7	D6	D5	D4	D3	D2	D1	D0	0	1	1
Byte 2 Register Select	1	0	0	0	0	RS2	RS1	RS0	0	1	1
WRITE TO VRAM											
Byte 1 Address set up	A7	A6	A5	A4	A3	A2	A1	A0	0	1	1
Byte 2 Address set up	0	1	A13	A12	A11	A10	A9	A8	0	1	1
Byte 3 Data Write	D7	D6	D5	D4	D3	D2	D1	D0	0	1	0
READ FROM VDP REGISTER											
Byte 1 Data Read	D7	D6	D5	D4	D3	D2	D1	D0	1	0	1
READ FROM VRAM											
Byte 1 Address set up	A7	A6	A5	A4	A3	A2	A1	A0	0	1	1
Byte 2 Address set up	0	0	A13	A12	A11	A10	A9	A8	0	1	1
Byte 3 Data Read	D7	D6	D5	D4	D3	D2	D1	D0	1	0	0

CPU READ FROM VDP STATUS REGISTER
The CPU can read the contents of the status register with a single-byte transfer. MODE is high for the transfer. CSR is used to signal the VDP that a read operation is required.

CPU READ FROM VRAM
The CPU reads data from VRAM through the VDP using the autoincrementing address register. A one-byte transfer is then required to read the data from the addressed VRAM byte. The address register is then autoincremented. Sequential VRAM data reads require only a one-byte transfer since the address register is already set up. During setup of the address register, the two most significant bits of the second address byte must be 0's. By setting up the address this way, a read cycle to VRAM is initiated and read data will be available for the first data transfer to the CPU. (See Table 1.) MODE is high for the address byte transfers and low for the data transfers. The VDP requires approximately 8 microseconds to fetch the VRAM byte following a data transfer and 3 microseconds following address setup.

VDP INTERRUPT
The VDP INT output pin is used to generate an interrupt at the end of each active display scan, which is about every 1/50 second (1/60 North America). The INT output is active when the interrupt Enable bit (IE) in VDP register 1 is a '1' and the F bit of the status register is a '1'. Interrupts are cleared when the status register is read.

VDP INITIALISATION
The VDP is externally initialised whenever the RESET input is active (low) and must be held low for a minimum of 3 microseconds. The external reset synchronises all clocks with its falling edge, sets the horizontal and vertical counters to known states, and clears VDP registers 0 and 1. The video display is automatically blanked since the BLANK bit in VDP register 1 becomes a '0'. The VDP, however, continues to refresh the VRAM even though the display is blanked. While the RESET line is active. the VDP does not refresh VRAM.

VDP/VRAM INTERFACE
The VDP can access up to 16,384 bytes of VRAM using a 14-bit VRAM address. The VDP fetches data from the VRAM in order to process the video image as described later. The VDP also stores data in or reads in data from the VRAM during a CPU-VRAM data transfer. The VDP automatically refreshes the VRAM.

VRAM INTERFACE CONTROL SIGNALS
The VDP-VRAM interface consists of two unidirectional 8-bit data buses and three control lines. The VRAM outputs data to the VDP on the VRAM read data bus (RD0-RD7). The VDP outputs both the address and data to the VRAM over the VRAM address/data bus (AD0-AD7). The VRAM row address is output when RAS is active (low). The column address is output when CAS is active (low). Data is output to the VRAM when R/W is active (low).

WRITE-ONLY REGISTERS
The eight VDP write-only registers are shown in Table 2. Registers 0 and 1 contain flags to enable or disable various VDP features and modes. Registers 2 through 6 contain values that specify starting locations of various sub-blocks of VRAM. Register 7 is used to define backdrop and text colours.

REGISTER	MSB<—Bit—>LSB 0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	M3	EV
1	4/16K	BLANK	IE	M1	M2	0	SIZE	MAG
2	0	0	0	0	NAME TABLE BASE ADDRESS			
3	<——COLOUR TABLE BASE ADDRESS——>							
4	0	0	0	0	0	PATTERN GENERATOR BASE ADDRESS		
5	0	<——SPRITE ATTRIBUTE TABLE BASE ADDRESS——>						
6	0	0	0	0	0	SPRITE PATTERN GENERATOR BASE ADDRESS		
7	<——TEXT COLOUR 1——>				TEXT COLOUR 0/BACKDROP COLOUR			
STATUS READ ONLY	F	5S	C	<——FIFTH SPRITE NUMBER——>				

TABLE 2. VDP REGISTERS

The following is a description of each register:

REGISTER 0 contains two VDP option control bits. All other bits are reserved for future use and must be '0's.

BIT 6 M3 (mode bit 3).
BIT 7 External Video enable/disable
'1' enables external video input
'0' disables external video input

REGISTER 1 contains 8 VDP option control bits.

BIT 0 4/16k selection
'0' selects 4K RAM operation
'1' selects 16K RAM operation (MTX operation)

BIT 1 BLANK enable/disable
'0' causes the active display area to blank
'1' enables the active display
Blanking causes the display to show border colour only

BIT 2 IE (Interrupt Enable)
'0' disable VDP interrupt
'1' enable VDP interrupt

BIT 3,4 M1, M2 (mode bits 1 and 2)
M1, M2 and M3 determine the operating mode of the VDP:

M1	M2	M3	
0	0	0	Graphics I mode
0	0	1	Graphics II mode
0	1	0	Multicolour mode
1	0	0	Text mode

BIT 5 Reserved

BIT 6 Size (sprite size select)
'0' selects Size 0 sprites (8 x 8 bits)
'1' selects Size 1 sprites (16 x 16 bits)

BIT 7 MAG (Magnification option for sprites)
'0' selects MAG0 sprites (1x)
'1' selects MAG1 sprites (2x)

REGISTER 2 defines the base address of the Name Table sub-block. The range on its contents is from 0 to 15. The contents of the register form the upper 4 bits of the 14-bit Name Table addresses; thus the Name Table base address is equal to (register 2) * 400h.

REGISTER 3 defines the base address of the Colour Table sub-block. The range on its contents is from 0 to 255. The contents of the register form the upper 8 bits of the 14-bit Colour Table addresses; thus the Colour Table base address is equal to (register 3) * 40h.

REGISTER 4 defines the base address of the Pattern, Text or Multicolour Generator sub-block. The range of its contents is 0 through 7. The contents of the register form the upper 3 bits of the 14-bit Generator addresses; thus the Generator base address is equal to (register 4) * 800h.

REGISTER 5 defines the base address of the Sprite Attribute Table sub-block. The range of its contents is from 0 through 127. The contents of the register form the upper 7 bits of the 14-bit Sprite Attribute Table addresses; thus the base address is equal to (register 5) * 80h.

REGISTER 6 defines the base address of the Sprite Pattern Generator sub-block. The range of its contents is 0 through 7. The contents of the register form the upper 3 bits of the 14-bit Sprite Pattern Generator addresses; thus the Sprite Pattern Generator base address is equal to (register 6) * 800h.

REGISTER 7 The upper 4 bits contain the colour code of colour 1 in the Text mode. The lower 4 bits contain the colour code for colour 0 in the Text mode and the backdrop colour in all modes. See Table 3 for colour codes.

STATUS REGISTER
The VDP has a single 8-bit status register that can be accessed by the CPU. The status register contains the interrupt pending flag, the sprite coincidence flag, the fifth sprite flag, and the fifth sprite number, if one exists. The format of the status register is shown in Table 2. A discussion of the contents follows.
The status register may be read at any time to test the F, C, and 5S status bits. Reading the status register will clear the interrupt flag, F. Asynchronous reads will, however, cause the frame flag (F) bit to be reset and therefore missed. Consequently, the status register should be read only when the VDP interrupt is pending.

INTERRUPT FLAG (F)
The F status flag in the status register is set to '1' at the end of the raster scan of the last line of the active display. It is reset to a '0' after the status register is read or when the VDP is externally reset. If the Interrupt Enable bit in VDP register 1 is active ('1'), the VDP interrupt output (INT) will be active (low) whenever the F status flag is a '1'.

COINCIDENCE FLAG (C)
The C status flag in the status register is set to a '1' if two or more sprites "coincide". Coincidence occurs if any two sprites on the screen have one or more overlapping pixels. Transparent coloured sprites, as well as those that are partially or completely off the screen, are also considered. Sprites beyond the Sprite Attribute Table terminator (D016) are not considered. The 'C' flag is cleared to a '0' after the status register is read or the VDP is externally reset.

FIFTH SPRITE FLAG (5S) AND NUMBER
The 5S status Flag in the status register is set to a '1' whenever there are five or more sprites on a horizontal line (lines 0 to 192) and the frame flag is equal to a '0'. The 5S status flag is cleared to a '0' after the status register is read or the VDP is externally reset. The number of the fifth sprite is placed into the lower 5 bits of the status register when the 5S flag is set and is valid whenever the 5S flag is '1'. The setting of the fifth sprite flag will not cause an interrupt.
The VDP operates at 262 lines per frame and approximately 60 frames per second in a non-interlaced mode of operation.

TABLE 3 - SCREEN DISPLAY PARAMETERS

PARAMETER	PIXEL CLOCK CYCLES	
HORIZONTAL	PATTERN/MULTICOLOUR	TEXT
Horizontal Active Display	256	240
Right Border	15	25
Right Blanking	8	8
Horizontal Sync	26	26
Left Blanking	2	2
Colour Burst	14	14
Left Blanking	8	8
Left Border	13	19
	342	342
VERTICAL	**LINE**	
Vertical Active Display	192	
Bottom Border	24	
Bottom Blanking	3	
Vertical Sync	3	
Top Blanking	13	
Top Border	27	
	262	

Video Display Modes

The VDP displays an image on the screen that can best be envisaged as a set of display planes sandwiched together. Figure 2 shows the definition of each of the planes. Objects on planes closest to the viewer have higher priority. In cases where two entities on two different planes are occupying the same spot on the screen, the entity on the higher priority plane will show at that point. For an entity on a specific plane to show through, all planes in front of that plane must be transparent at that point. The first 32 planes each may contain a single sprite. (Sprites are pattern objects whose positions on the screen are defined by horizontal and vertical co-ordinates in VRAM.) The areas of the Sprite Planes, outside the sprite itself, are transparent. Since the co-ordinates of the sprite are in terms of pixels, the sprite can be positioned and moved about very accurately. Sprites are available in three sizes: 8 X 8 pixels, 16 X 16 pixels, and 32 X 32 pixels. Behind the Sprite Plane is the Pattern Plane. The Pattern Plane is used for textual and graphics images generated by the Text, Graphics I, Graphics II, or Multicolour modes. Behind the Pattern Plane is the backdrop, which is larger in area than the other planes so that it forms a border around them. The last and lowest priority plane is the External Video Plane. Its image is defined by the external video input pin. The backdrop consists of a single colour used for the display borders and as the default colour for the active display area. The default colour is stored in the VDP register 7. When the backdrop colour register contains the transparent code, the backdrop automatically defaults to black if the external video mode is not selected.

The 32 Sprite Planes are used for the 32 sprites in the Multicolour and Graphics modes. They are not used in the Text mode and are automatically transparent. Each of the sprites can cover an 8 X 8, 16 X 16, or 32 X 32 pixel area on its plane. Any part of the plane not covered by the sprite is transparent. All or part of each sprite may also be transparent. Sprite 0 is on the outside or highest plane, and sprite 31 is on the plane immediately adjacent to the Pattern Plane. Whenever a pixel in a Sprite Plane is transparent, the colour of the next plane can be seen through that plane. If, however, the sprite pixel is non-transparent, the colours of the lower planes are automatically replaced by the sprite colour. There is also a restriction on the number of sprites on a line. Only four sprites can be active on any horizontal line. Additional sprites on a line will be automatically made transparent for that line. Only those sprites that are active on the display will cause the coincidence flag to set. The VDP status register provides a flag bit and the number of the fifth sprite whenever this occurs. The Pattern Plane is used in the Text, Multicolour, and Graphics modes for display of the graphic patterns of characters. Whenever a pixel on the Pattern Plane is non-transparent, the backdrop colour is automatically replaced by the Pattern Plane colour. When a pixel in the Pattern Plane is transparent, the backdrop colour can be seen through the Pattern Plane.

FIGURE 2. VDP DISPLAY PLANES

The VDP has four video colour display modes that appear on the Pattern Plane: Graphics I mode, Graphics II mode, Text mode, and Multicolour mode. Graphics I and Graphics II modes cause the Pattern Plane to be broken up into groups of 8 X 8 pixels, called pattern positions. Since the full image is 256 X 192 pixels, there are 32 X 24 pattern positions on the screen in the graphics modes. In Graphics I mode, 256 possible patterns may be defined for the 768 pattern positions with two unique colours allowed for each pattern definition. Graphics II mode provides, through a unique mapping scheme, 768 pattern definitions for the 768 pattern positions. Graphics II mode also allows the selection of two unique colours for each line of a pattern definition. Thus, all 15 colours plus transparent may be used in a single pattern position. In Text mode, the Pattern Plane is broken into groups of 6 X 8 pixels, called text positions. There are 40 X 24 text positions on the screen in this mode. In Text mode, sprites do not appear on the screen and two colours are defined for the entire screen. In Multicolour mode, the screen is broken into a grid of 64 X 48 positions, each of which is a 4 X 4 pixel. Within each position, one unique colour is allowed.

The VDP registers define the base addresses for several sub-blocks within VRAM. These sub-blocks form tables which are used to produce the desired image on the TV screen. The Pattern Name Table, the Pattern Generator Table and the Sprite Generator Table are used to form the sprites. The contents of these tables must all be provided by the microprocessor. Animation is achieved by altering the contents of VRAM in real time.

The VDP can display the 15 colours, plus transparent shown in Table 3. The VDP colours also provide eight different grey levels for displays on monochrome televisions; the luminance values in the table indicate these levels, 0.00 being black and 1.00 being white. Whenever all planes are of the transparent colour at a given point, the colour shown at that point will be black.

TABLE 4. Colour Assignments

COLOUR (HEX)	COLOUR	LUMINANCE (DC VALUE)	CHROMINANCE (AC VALUE)
0	Transparent	0.00	-
1	Black	0.00	-
2	Medium Green	.60	.60
3	Light Green	.80	.53
4	Dark Blue	.47	,73
5	Light Blue	.67	.60
6	Dark Red	.53	.53
7	Cyan	.80	.73
8	Medium Red	.67	.73
9	Light Red	.80	.73
A	Dark Yellow	.87	.53
B	Light Yellow	1.00	.40
C	Dark Green	.47	.60
D	Magenta	.60	.47
E	Grey	-	-
F	White	1.00	-

Graphics 1 Mode

The VDP is in Graphics 1 mode when M1, M2, and M3 bits in VDP registers 1 and 0 are zero. In Graphics 1 mode the Pattern Plane is divided into a grid of 32 columns by 24 rows of pattern positions. Each of the pattern positions contains 8 x 8 pixels. The table in VRAM is used to generate the Pattern Plane. A total of 2848 VRAM bytes are required for the Pattern Name, Colour and Generator tables. Less memory is required if all 256 possible pattern definitions are not required. The tables can be overlapped to reduce the amount of VRAM needed for pattern generation.

The Pattern Generator Table contains a library of patterns that can be displayed in the pattern positions. It is 2048 bytes long, and is arranged into 256 patterns, each of which is eight bytes long, yielding 8 x 8 bits. All of the '1's in the eight-byte pattern can designate one colour (colour 1), while all the '0's can designate another colour (colour 0).

The full 8-bit pattern name is used to select one of the 256 pattern definitions in the Pattern Generator Table. The table is a 2048-byte block in VRAM beginning on a 2 kilobyte boundary. The starting address of the table is determined by the generator base address in VDP register 4. The base address forms the three most significant bits of the 14-bit VRAM address for each Pattern Generator Table entry. The next 8 bits indicate the 8-bit name of the selected pattern definition. The lowest 3 bits of the VRAM address indicate the row number within the pattern definition.

Eight bytes are required for each of the 256 possible unique 8 x 8 pattern definitions. The first byte defines the first row of the pattern, and the second byte defines the second row. The first bit of each of the eight bytes define the first column of the pattern. The remaining rows and columns are similarly defined. Each bit entry in the pattern definition selects one of the two colours for that pattern. A '1' bit selects the colour code (colour 1) contained in the most significant four bits of the corresponding colour table byte. A '0' bit selects the other colour code (colour 0). An example of pattern definition mapping is provided below.

Row/byte	Column						Bit							
	0	1	2	3	4	5	0	1	2	3	4	5	6	7
0		*	*	*	*	*	0	1	1	1	1	1	0	0
1						*	0	0	0	0	0	1	0	0
2			*	*	*	*	0	0	1	1	1	1	0	0
3						*	0	0	0	0	0	1	0	0
4						*	0	0	0	0	0	1	0	0
5						*	0	0	0	0	0	1	0	0
6		*	*	*	*	*	0	1	1	1	1	1	0	0
7							0	0	0	0	0	0	0	0
		——PATTERN——						——PATTERN DEFINITION——						

The colour of the '1's and '0's is defined by the Pattern Colour Table that contains 32 entries each of which is one byte long. Each entry defines two colours: the most significant 4 bits of each entry define the colour of the '1's, and the least significant 4 bits define the colour of the '0's. The first entry in the colour table defines the colours for patterns 0 to 7; the next entry for patterns 8 to 15, and so on. (See Table 4 for assignments.) Thus, 32 different pairs of colours may be displayed simultaneously.

The Pattern Name Table is located in a contiguous 768-byte block in VRAM beginning on a 1 kilobyte boundary. The starting address of the Name Table is determined by the 4-bit Name Tablebase address field in VDP register 2. The base address forms the upper four bits of the 14-bit VRAM address. The lower 10 bits of the VRAM address are formed from the row and column counters.

Each byte entry in the Name Table is the name of or the pointer to a pattern definition in the Pattern Generator Table. The upper five bits of the eight-bit name identify the colour group of the pattern. There are 32 groups of eight patterns. The same two colours are used for all eight patterns in a group; the colour codes are stored in the VDP Colour Table. The Colour Table is located in a 32-byte block in VRAM beginning on a 64-byte boundary. The table starting address is determined by the 8-bit Colour Tablebase address in VDP register 3. The base address forms the upper eight bits of the 14-bit Colour Table entry VRAM address. The next bit is a '0' and the lowest 5 bits are equal to the upper 5 bits of the corresponding Name Table entries.

Since the tables in VRAM have their base addresses defined by the VDP registers, a complete switch of the values in the tables can be made by simply changing the values in the VDP registers. This is especially useful when one wishes to time slice between two or more screens of graphics.

When the Pattern Generator Table is loaded with a pattern set, manipulation of the Pattern Name Table contents can change the appearance of the screen. Alternatively, a dynamically changing set of patterns throughout the course of a graphics session is easily accomplished since all tables are in VRAM.

For textual applications, the desired character set is typically loaded into the Pattern Generator first. The official US ASCII character set might be loaded into the Pattern Generator in such a way that the pattern numbers correspond to the 8-bit ASCII codes for that pattern; e.g., the pattern for the letter "A" would be loaded into pattern number 4116 in the Pattern Generator. Next the Pattern Colour Table would be loaded up with the proper colour set. To print a textual message on the screen, write the proper ASCII codes out to the Pattern Name Table.

Images can be formed using the Pattern Plane. To display an object of size 8 x 8 pixels or smaller, only one pattern would need to be defined. To display a larger figure, the figure should be broken up into smaller 8 x 8 squares. Then multiple patterns can be defined, and the Pattern Generator and Pattern Name Table set up appropriately. Note that rough motion of objects requires merely updating entries in the Pattern Name Table.

TABLE 5 Pattern colour table

BYTE No.	PATTERN No.
0	0..7
1	8..15
2	16..23
3	24..31
4	32..39
5	40..47
6	48..55
7	56..63
8	64..71
9	72..79
10	80..87
11	88..95
12	96..103
13	104..111
14	112..119
15	120..127
16	128..135
17	136..143
18	144..151
19	152..159
20	160..167
21	168..175
22	176..183
23	184..191
24	192..199
25	200..207
26	208..215
27	216..223
28	224..231
29	232..239
30	240..247
31	248..255

A total of 2848 VRAM bytes are required for the Pattern, Name, Colour and Generator tables. Less memory is needed if all 256 possible pattern definitions are not required; the tables can be overlapped to reduce the amount of VRAM needed for pattern generation.

Graphics II Mode

The VDP is in the Graphics II mode when mode bits M1 = 0, M2 = 0, and M3 = 1. The Graphics II mode is similar to Graphics I mode except it allows a larger library of patterns so that a unique pattern generator entry may be made for each of the 768 (32 x 24) pattern positions on the video screen. Additionally, more colour information is included in each 8 x 8 graphics pattern. Thus two unique colours may be specified for each byte of the 8 x 8 pattern. A larger amount of VRAM (12 kilobytes) is required to implement the full usage of the Graphics II mode.

Like Graphics I mode, the Graphics II mode Pattern Name Table contains 768 entries which correspond to the 768 pattern positions on the display screen. Because the Graphics 1 mode pattern names are only 8 bits in length, a maximum of 256 pattern definitions may be addressed using the addressing scheme discussed in the previous section. Graphics II mode, however, segments the display screen into three equal parts of 256 pattern positions each, and also segments the Pattern Generator Table into three equal blocks of 2048 bytes each. Pattern definitions in the first third correspond to pattern positions in the upper third of the display screen. Likewise pattern definitions in the second and third blocks of the Pattern Generator Table correspond to the second and third areas of the Pattern Plane. The Pattern Name Table is also segmented into three blocks of 256 names each so that names found in the upper third, reference pattern definitions found in the upper 2048 bytes in the Pattern Generator Table. Likewise the second and third blocks reference pattern definitions in the second 2048 byte block and third 2048 byte block respectively. Thus, if 768 patterns are uniquely specified an 8-bit pattern name will be used three times, once in each segment of the Pattern Name Table. The Pattern Generator Table falls on eight kilobyte boundaries and may be located in the upper or lower half of 16K memory based on the MSB of the pattern generator base in VDP register 4. The LSB's must be set to all '1's.

The Colour Table is also 6144 bytes long and is segmented into three equal blocks of 2048 bytes. Each entry in the Pattern Colour Table is eight bytes which provides the capability to uniquely specify colour 1 and colour 0 for each of the eight bytes of the corresponding pattern definition. The addressing scheme is exactly like that of the Pattern Generator Table except for the location of the table in VRAM. This is controlled by the loading of the MSB of the colour base in VDP register 3. The LSB's must be set to all '1's.

Multicolour Mode

The VDP is in Multicolour mode when mode bits M1 = 0, M2 = 1, and M3 = 0. Multicolour mode provides an unrestricted 64 x 48 colour square display. Each colour square contains a 4 x 4 block of pixels. The colour of each of the colour squares can be any one of the 15 video display colours plus transparent. Consequently, all 15 colours can be used simultaneously in the Multicolour mode. The Backdrop and Sprite Planes are still active in the Multicolour mode.

The Multicolour Name Table is the same as that for the graphics modes, consisting of 768 name entries. The name no longer points to a colour list; rather colour is now derived from the Pattern Generator Table. The name points to an eight-byte segment of VRAM in the Pattern Generator Table.

Only two bytes of the eight-byte segment are used to specify the screen image. These two bytes specify four colours, each colour occupying a 4 x 4 pixel area. The four MSB's of the first byte define the colour of the upper left quarter of the multicolour pattern; the LSB's define the colour of the upper right quarter. The second byte similarly defines the lower left and right quarters of the multicolour pattern. The two bytes thus map into a 8 x 8 pixel multicolour pattern.

The location of the two bytes within the eight-byte segment pointed to by the name is dependent upon the screen position where the name is mapped. For names in the top row (names 0-31), the two bytes are the first two within the groups of eight-byte segments pointed to by the names. The next row of names (32-63) uses the third and fourth bytes within the eight-byte segments. The next row of names uses the fifth and sixth bytes while the last row of names uses the seventh and eighth. This series repeats for the remainder of the screen.

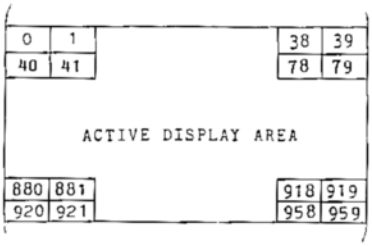
The mapping of VRAM contents to screen image is simplified by using duplicate names in the Name Table. Since the series of bytes used within the eight-byte segment repeats every four rows, the four rows in the same column can use the same name. Then the eight-byte segment specifies a 2 x 8 colour square pattern on the screen as a straightforward translation from the eight-byte segment in VRAM pointed to by the common name.

When used in this manner, 768 bytes are still used for the Name Table and 1536 bytes are used for the colour information in the Pattern Generator Table (24 rows x 32 columns x 8 bytes/pattern position). Thus a total of 1728 bytes in VRAM are required. It should be noted that the tables begin on even 1K and 2K boundaries and are therefore not contiguous.

Text Mode

The VDP is in Text mode when mode bits M1 = 1, M2 = 0, and M3 = 0. In the Text mode, the screen is divided into a grid of 40 text positions across and 24 down. Each of the text positions contains six pixels across and eight pixels down. The tables used to generate the Pattern Plane are the Pattern Name Table and the Pattern Generator Table. There can be up to 256 unique patterns defined at any time. The pattern definitions are stored in the Pattern Generator Table in VRAM and can be dynamically changed. The VRAM contains a Pattern Name Table which maps the pattern definitions into each of the 960 pattern cells on the Pattern Plane. Sprites are not available in Text mode.

TEXT MODE NAME TABLE PATTERN POSITIONS



As in the case of the Graphics modes, the Pattern Generator Table contains a library of text patterns that can be displayed in the text positions. It is 2048 bytes long, and is arranged in 256 text patterns, each of which is eight bytes long. Since each text position on the screen is only six pixels across, the least significant 2 bits of each text pattern are ignored, yielding 6 x 8 bits in each text pattern. Each block of eight bytes defines a text pattern in which all the '1's in the text pattern take on one colour when displayed on the screen, while all the '0's take on another colour. These colours are chosen by loading VDP register 7 with the colour 1 and colour 0 in the left and right nibbles respectively.

In the Text mode, the Pattern Name Table determines the position of the text pattern on the screen. There are 960 entries in the Pattern Name Table, each one byte long. There is a one-to-one correspondence between text pattern positions on the screen and entries in the Pattern Name Table (40*24 = 960). The first 40 entries correspond to the top row of text pattern positions on the screen, the next forty to the second row, and so on. The value of an entry in the Pattern Name Table indicates which of the 256 text patterns is to be placed at that spot on the Pattern Plane. The Pattern Name Table is located in a contiguous 960-byte block in VRAM beginning on a 1 kilobyte boundary. The starting address of the name table is determined by the 4-bit Name Table base address field in VDP register 2. The base address forms the upper 4 bits of the 14-bit VRAM address. The lower 10 bits of the VRAM address point to one of 960 pattern cells. The name table is organised by rows. Each byte entry in the name table is the pointer to a pattern definition in the Pattern Generator Table. The same two colours are used for all 256 patterns; the colour codes are stored in VDP register 7.

As its name implies, the Text mode is intended mainly for textual applications, especially those in which the 32 patterns per line in Graphics modes is insufficient. The advantage is that eight more patterns can be fitted onto one line; the disadvantages are that sprites cannot be used, and only two colours are available for the entire screen. With care, the same text pattern set that

is used in Text mode can be also used in Graphics 1 mode. This is done by ensuring that the least significant 2 bits of all the character patterns are '0'. A switch from Text mode to Pattern mode, then, results in a stretching of the space between characters, and a reduction of the number of characters per line from 40 to 32. As with the Graphics Modes, once a character set has been defined and placed into the Pattern Generator, updating the Pattern Name Table will produce and manipulate textual material on the screen.

The full 8-bit pattern name is used to select one of the 256 pattern definitions in the pattern generator table. The table is a 2048-byte block in VRAM beginning on a 2 kilobyte boundary. The starting address of the table is determined by the generator base address in VDP register 4. The base address forms the 3 most significant bits of the 14-bit VRAM address for each Pattern Generator Table entry. The next 8 bits are equal to the 8-bit name of the selected pattern definition. The lowest 3 bits of the VRAM address are equal to the row number within the pattern definition.

Eight bytes are required for each of the 256 possible unique 6 x 8 pattern definitions. The first byte defines the first row of the pattern, and the second byte defines the second row. The two least significant bits in each byte are not used. It is, however, strongly recommended that these bits be '0's. Each bit entry in the pattern definition selects one of the two colours for that pattern. A '1' bit selects the colour code (colour 1) contained in the most significant 4 bits of VDP register 7. A '0' bit selects the other colour code (colour 0) which is in the least significant 4 bits of the same VDP Register.

A total of 3005 VRAM bytes are required for the Pattern Name and Generator Tables. Less memory is required if all 256 possible pattern definitions are not required; the tables can be overlapped to reduce the amount of VRAM needed for pattern generation.

Sprites

The video display can have up to 32 sprites on the highest priority video planes. The sprites are special animation patterns which provide smooth motion and multilevel pattern overlaying. The location of a sprite is defined by the top left hand corner of the sprite pattern. The sprite can be easily moved pixel by pixel by redefining the sprite origin. This provides a simple but powerful method of quickly and smoothly moving special patterns. The sprites are not active in the Text mode. The 32 Sprite Planes are fully transparent outside of the sprite itself.

The sub-blocks in VRAM that define sprites are the Sprite Attribute Table and the Sprite Generator Table. These tables are similar to their equivalents in the pattern realm in that the Sprite Attribute Table specifies where the sprite appears on the screen, while the Sprite Generator Table describes what the sprite looks like, Sprite Pattern formats are given in Table 5.

Since there are 32 sprites available for display, there are 32 entries in the Sprite Attribute Table. Each entry consists of four bytes. The entries are ordered so that the first entry corresponds to the sprite on the sprite 0 plane, the next to the sprite on the sprite 1 plane, and so on. The Sprite Attribute Table is 4*32 = 128 bytes long. The Sprite Attribute Table is located in a contiguous 128-byte block in VRAM beginning on a 128-byte boundary. The starting address of the Attribute Table is determined by the Sprite Attribute Table base address in VDP register 5. The base address forms the upper seven bits of the 14 bit VRAM address. The next 5 bits of the VRAM address are equal to the sprite number. The lowest 2 bits select one of the four bytes in the Attribute Table entry for each sprite. Each Sprite Attribute Table entry contains four bytes which specify the sprite position, sprite pattern name, and colour.

TABLE 6 Sprite pattern formats

SIZE	MAG	AREA	RESOLUTION	BYTES/PATTERN
0	0	8x8	single pixel	8
1	0	16x16	single pixel	32
0	1	16x16	2x2 pixels	8
1	1	32x32	2x2 pixels	32

The first two bytes of each entry of the Sprite Attribute Table determine the position of the sprite on the display. The first byte indicates the vertical distance of the sprite from the top of the screen, in pixels. It is defined such that a value of -1 puts the sprite butted up at the top of the screen, touching the backdrop area. The second bytes describes the horizontal displacement of the sprite from the left edge of the display. A value of 0 butts the sprite up against the left edge of the backdrop. Note that it is from the upper left pixel of the sprite that all measurements are taken.

When the first two bytes of an entry position of a sprite are overlapping the backdrop, the part of the sprite that is within the backdrop is displayed normally. The part of the sprite that overlaps the backdrop is hidden from view by the backdrop. This allows the animator to move a sprite into the display from behind the backdrop. The displacement in the first byte is partially signed, in that values for vertical displacement between -31 and 0 (E116 to 0) allow a sprite to "bleed in" from the top edge of the backdrop. Likewise, values in the range of 207 to 191 allow the sprite to bleed in from the bottom edge of the backdrop. Similarly, horizontal displacement values in the vicinity of 255 allow a sprite to bleed in from the right side of the screen. To allow sprites to bleed in from the left edge of the backdrop, a special bit in the third byte of the Sprite Attribute Table entry is used, as described in a later paragraph.

Byte 3 of the Sprite Attribute Table entry contains the pointer to the Sprite Generator Table that specifies what the sprite should look like. This is an 8-bit pointer to the sprite patterns definition, the Sprite Generator Table. The sprite name is similar to that in the Patterns Graphic mode.

Byte 4 of the Sprite Attribute Table entry contains the colour of the sprite in its lower 4 bits (see Table 2 for colour codes). The most significant bit is the Early Clock bit (EC). This bit, when set to a '0', does nothing. When set to '1', the horizontal position of the sprite is shifted to the left by 32 pixels. This allows a sprite to bleed in from the left edge of the backdrop. Values for horizontal displacement (byte 2 in the entry) in the range 0 to 32 cause the sprite to overlap with the left hand border of the backdrop.

The Sprite Generator Table is a maximum of 2048 bytes long beginning on the 2 kilobyte boundaries. It is arranged into 256 blocks of 8 bytes each. The third byte of the Sprite Attribute Table entry, then, specifies which eight byte block to use to specify a sprite's shape. The '1's in the Sprite Generator cause the sprite to be defined at that point;'0's cause the transparent colour to be used. The starting address of the table is determined by the sprite generator base address in the VDP register 6. The base address forms the 3 most significant bits of the 14-bit VRAM address. The next 8 bits of the address are equal to sprite name, and the last 3 bits are equal to the row number within the sprite pattern. The address formation is slightly modified for SIZE 1 sprites.

There is a maximum limit of four sprites that can be displayed on one horizontal line. If this rule is violated, the four highest-priority sprites on the line are displayed normally. The fifth and subsequent sprites are not displayed on that line. Furthermore, the fifth sprite bit in the VDP status register is set to a '1', and the number of the violating fifth sprite is loaded into the status register.

Larger sprites than 8x8 pixels can be used if desired. The MAG and SIZE bits in VDP register 1 are used to select the various options. The options are described here:

MAG=0,SIZE=0:	No options chosen
MAG=1,SIZE=0:	Eight bytes are still used in the Sprite Generator Table to describe the sprite; however, each bit in the Sprite Generator maps into 2 x 2 pixels on the TV screen, effectively doubling the size of the sprite to 16 x 16.
MAG=0,SIZE=1:	31 bytes are used in the Sprite Generator Table to define the sprite shape; the result is a 16 x 16 pixel sprite. Mapping is still one-bit-to-one pixel.
MAG=1,SIZE=1:	Same as MAG=0,SIZE=1 except each bit now maps into a 2 x 2 pixel area, yielding a 32 x 32 sprite.

The VDP provides sprite coincidence checking. The coincidence status flag in the VDP status register is set to a '1' whenever two active sprites have '1' bits at the same screen location.

Sprite processing is terminated if the VDP finds a value of 208 (D016) in the vertical position field of any entry in the Sprite Attribute Table. This permits the Sprite Attribute Table to be shortened to the minimum size required; it also permits the user to blank out part or all of the sprites by simply changing one byte in VRAM.

A total of 2176 VRAM bytes are required for the Sprite Name and Pattern Generator Tables. Significantly less memory is required if all 256 possible sprite pattern definitions are not required. The Sprite Attribute Table can also be shortened as described above. The tables can be overlapped to reduce the amount of VRAM required for sprite generation.

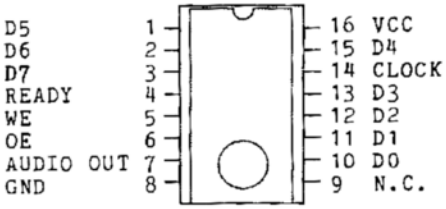
7 SOUND GENERATOR

The sound processor used in the MTX500 Series computers is the Texas Instruments SN76489A sound generator IC. This device is I/O mapped as follows:

Data is mapped to output port 6
Strobe line is mapped to input port 3

To write data to the device send valid data to output port 6 and then strobe the data into the device by performing a dummy read from input port 3. The time interval between successive reads must be at least 32 clock cycles (32 T-states).

SN76489A Pin-Out (Top view)



DESCRIPTION

The SN76489A digital complex sound generator is an I^2 L/Bipolar IC designed to provide low cost tone/noise generation capability in microprocessor systems. The SN76489A is a data bus based I/O peripheral.

RECOMMENDED OPERATING CONDITIONS

PARAMETER	MIN	TYP	MAX	UNITS
Supply Voltage, VCC	4.5	5.0	5.5	V
High Level Output Voltage, VOH (pin 4)			5.5	V
Low Level Output Current, IOL (pin 4)			2	mA
Operating Free-Air Temperature,TA	0		70	oC

OPERATION

1 Tone Generators

Each tone generator consists of a frequency synthesis and an attenuation section. The frequency synthesis section requires 10 bits of information (F9-F0) to define half the period of the desired frequency (n). F9 is the most significant bit and F0 is the least significant bit. This information is loaded into a 10 stage tone counter, which is decremented at a N/16 rate where N is the input clock frequency. When the tone counter decrements to zero, a borrow signal is produced. This borrow signal toggles the frequency flip-flop and also reloads the tone counter. Thus, the period of the desired frequency is twice the value of the period register.

The frequency can be calculated by the following:

$$f = \frac{N}{32n}$$

where N = ref clock in Hz
n = 10 bit binary number

The output of the frequency flip-flop feeds into a four stage attenuator. The attenuator values, along with their bit position in the data word, are shown in Table 1. Multiple attentuation control bits may be true simultaneously. Thus, the maximum attenuation is 28 db.

TABLE 1 Attenuation Control

BIT POSITION				
A3	A2	A1	A0	WEIGHT
0	0	0	1	2 db
0	0	1	0	4 db
0	1	0	0	8 db
1	0	0	0	16 db
1	1	1	1	OFF

2 Noise Generator

The Noise Generator consists of a noise source and an attenuator. The noise source is a shift register with an exclusive OR feedback network. The feedback network has provisions to protect the shift register from being locked in the zero state.

TABLE 2 Noise Feedback Control

FB	CONFIGURATION
0	"Periodic" Noise
1	"White" Noise

Whenever the noise control register is changed, the shift register is cleared. The shift register will shift at one of four rates as determined by the two NF bits. The fixed shift rates are derived from the input clock.

TABLE 3 Noise Generator Frequency Control

BITS		
NF1	NF0	SHIFT RATE
0	0	N/512
0	1	N/1024
1	0	N/2048
1	1	Tone Generator "3 Output

The output of the noise source is connected to a programmable attenator as shown in figure 4.

3 Output Buffer/Amplifier
The output buffer is a conventional operational amplifier summing circuit. It sums the three tone generator outputs, and the noise generator output. The output buffer will generate up to 10mA.

4 CPU to SN76489A Interface
The microprocessor interfaces with the SN76489A by means of the 8 data lines and 3 control lines (WE, CE and READY). Each tone generator requires 10 bits of information to select the frequency and 4 bits of information to select the attenuation. A frequency update requires a double byte transfer, while an attenuator update requires a single byte transfer.

If no other control registers on the chip are accessed, a tone generator may be rapidly updated by initially sending both bytes of frequency and register data, followed by just the second byte of data for succeeding values. The register address is latched on the chip, so the data will continue going into the same register. This allows the 6 most significant bits to be quickly modified for frequency sweeps.

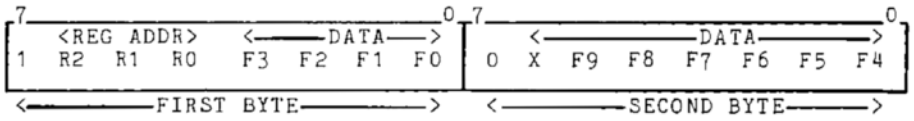
5 Control Registers
The SN76489A has 8 internal registers which are used to control the 3 tone generators and the noise source. During all data transfers to the SN76489A, the first byte contains a three bit field which determines the destination control register. The register address codes are shown in Table 4.

TABLE 4 Register Address Field

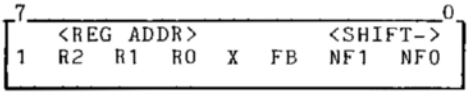
R2	R1	R0	DESTINATION CONTROL REGISTER
0	0	0	TONE 1 FREQUENCY
0	0	1	TONE 1 ATTENUATION
0	1	0	TONE 2 FREQUENCY
0	1	1	TONE 2 ATTENUATION
1	0	0	TONE 3 FREQUENCY
1	0	1	TONE 3 ATTENUATION
1	1	0	NOISE CONTROL
1	1	1	NOISE ATTENUATION

6 Data Formats
The formats required to transfer data are shown below.

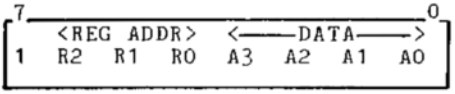
Update Frequency (Two Byte Transfer)



Update Noise Source (Single Byte Transfer)



Update Attenuator (Single Byte Transfer)



7 Data Formats
The microprocessor selects the SN76489A by placing CE into the true state (low voltage). Unless CE is true, no data can occur. When CE is true, the WE signal strobes the contents of the data bus to the appropriate control register. The data bus contents must be valid at this time.

The SN76489A requires approximately 32 clock cycles to load the data into the control register. The open collector READY output is used to synchronize the microprocessor to this transfer and is pulled to the false state (low voltage) immediately following the leading edge of CE. It is released to go to the true statement (external pullup) when the data transfer is completed.
The data transfer timing is shown below.

FIGURE 1. DATA TRANSFER TIMING

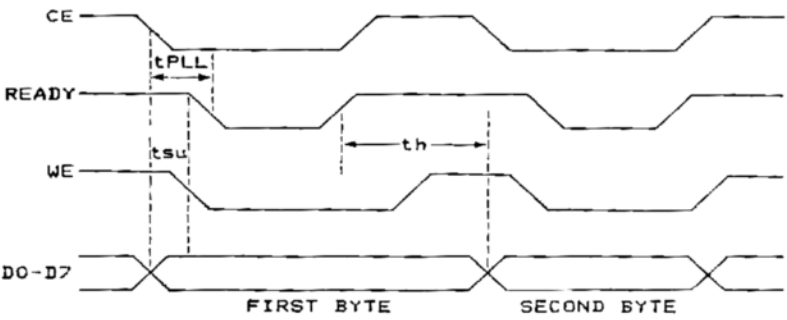
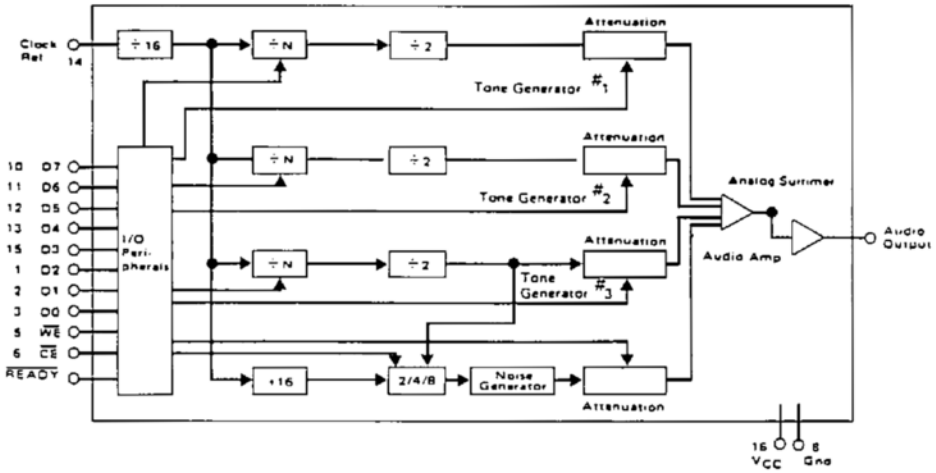


TABLE 5 Function Table

INPUTS		OUTPUT
CE	WE	READY
L	L	L
L	H	L
H	L	H
H	H	H

*This table is valid when the device is:
(1) not being clocked,
(2) is initialized by pulling WE and CE high.

SN76489A BLOCK DIAGRAM



BLOCK DIAGRAM DESCRIPTION

This device consists of three programmable tone generators, a programmable noise generator, a clock scaler, individual generator attentuators and an audio summer output buffer. The SN76489A has a parallel 8 bit interface through which the microprocessor transfers the data which controls the audio output.

8 MTX 500 SERIES MEMORY MAP

The paged memory map structure of the MTX Series computers has been designed to operate in two modes.

1 ROM BASED (RELCPMH = 0)

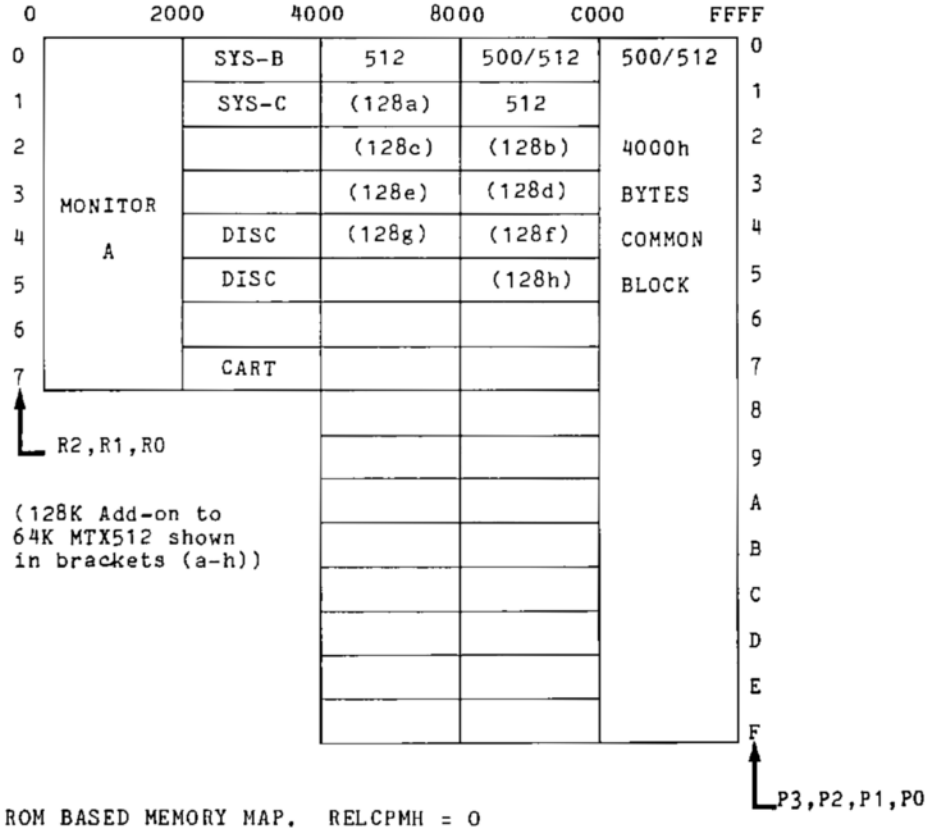
ROMs are mapped from 0 to 3FFFh. The 8K (2000h bytes) monitor ROM is always available in area 0 to 1FFFh and the paged ROMs of 8K (2000h bytes) each are mapped from 2000h to 3FFFh as eight pages 0 to 7 set by R2,R1,R0 in the page port write only register. Up to 512K of RAM is mapped on 16 pages (0 to F) set up by P3,P2,P1 and P0 in the page port write only register. The area C000h to FFFFh is a 16K (4000h bytes) block common to all RAM pages. The 32K (8000h bytes) block from 4000h to BFFFh is mapped as 16 pages. The 32K bytes of RAM for an MTX500 is mapped from 8000h to FFFFh (page 0). The 64K bytes of RAM for an MTX512 is mapped from 4000h to FFFFh (page 0). The additional 16K is mapped from 8000h to C000h on page 1.

2 RAM BASED (RELCPMH = 1)

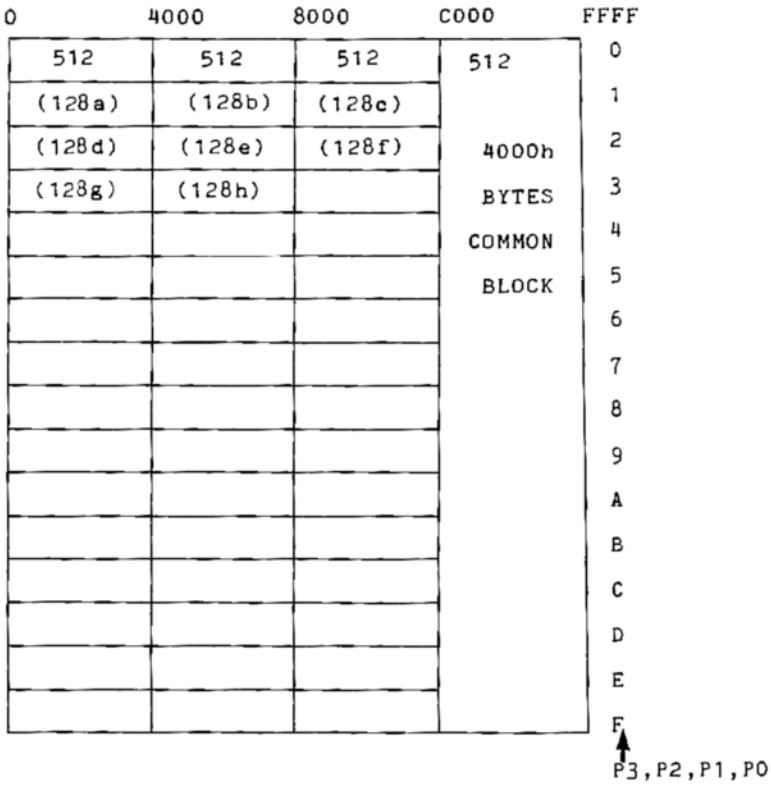
All ROMs are switched out in this mode, and up to 16 pages of 48K (C000h bytes) are mapped from 0 to BFFFh. These pages are set by P3,P2,P1 and P0 in the write only page port register. In the area C000h to FFFFh is a 16K block (4000h bytes) of RAM common to all pages.

D7	D6	D5	D4	D3	D2	D1	D0
RELCPMH	R2	R1	R0	P3	P2	P1	P0

Write only page port register, output port 0.



ROM BASED MEMORY MAP. RELCPMH = 0



RAM BASED MEMORY MAP. RELCPMH = 1
(128K Add-on to 64K MTX512 shown in brackets (a-h)

9 INPUT/OUTPUT PORT SUMMARY

This section describes the MTX Series Port Map

00h
INPUT
IN(0) is used to set the printer STROBE (active low) to LOW. The STROBE line is reset HIGH either on CPU RESET or by IN(4). In the event of interrupt while STROBE is low it would be good practice to reset STROBE within an interrupt routine extending over a period of more than a few microseconds.

OUTPUT
OUT(0),d defines memory page address. The bit map is as follows:

```
D0 = P0
D1 = P1
D2 = P2
D3 = P3
D4 = R0
D5 = R1
D6 = R2
D7 = RELCPMH
```

Where the nibble P(i) defines the RAM page address, the 3 bit R(i) defines the ROM page address and bit 7 defines a ROM based system (D7 = 0) or a RAM based system (D7 = 1). The latch is reset to 0 on CPU reset.

01h
INPUT
IN(1),d VDP read (mode = 0) together with port 02 provide two contiguous read/write ports for the VDP. See documentation on the TMS9918 Series. Note Z80 CPU address line A1 is connected to mode input.

OUTPUT
OUT(1),d VDP write (mode = 0).

02h
INPUT
IN(2),d VDP read (mode = 1)

OUTPUT
OUT(2),d VDP write (mode = 1)

03h
INPUT
IN(3) This line is used as an output strobe into the sound generator. After data has been latched into the output port (6) data may be immediately strobed in using this line. A total of at least 32 clock cycles must have elapsed before additional data may be strobed in using IN(3).

OUTPUT
OUT(3),d This is the cassette output serial line. Valid data is placed on D0. This data bit is latched and appears on the cassette output (MIC) after attenuation (-20dB*VCC) and low pass filtering.

04h
INPUT
IN(4),d This is a nibble port for monitoring the status of the Centronics type parallel printer port.

D0 = BUSY active high handshake line
D1 = ERROR active low
D2 = PE paper empty active high
D3 = SLCT printer in selected state active high

OUTPUT
OUT(4),d Parallel 8 bit printer data. Valid data should be latched into this port. When status on IN(4) reads not BUSY and selected, then data should be strobed after a delay of approximately 1 microsecond using IN(0) to force STROBE low. After a further delay of approximately 1 microsecond STROBE should be forced high using IN(4).

05h
INPUT
IN(5),d This port is used to read the least significant 8 bits from the ten bit sense line of the 8x10 keyboard matrix.

OUTPUT
OUT(5),d This latched port provides the 8 drive lines of the 8x10 keyboard matrix.

06h
INPUT
IN(6),d This port is used to read in the two most significant sense lines (D0 and D1) of the 8x10 keyboard matrix. The two bit country code switch is read on D2 and D3.

OUTPUT
OUT(6),d This port is used to provide latched data for the sound generator which is subsequently strobed using IN(3).

07h
INPUT
IN(7),d This is the input port for the uncommitted parallel input output port (PIO). Data may be latched in for reading with an active low pulse on the enable line, designated INSTB.

OUTPUT
OUT(7),d This is the output port of the PIO. It is a latched output with tri-state output control using OTSTB.

08,09,0A,0Bh
These are four contiguous read/write ports for the four channels of the Z80A CTC.

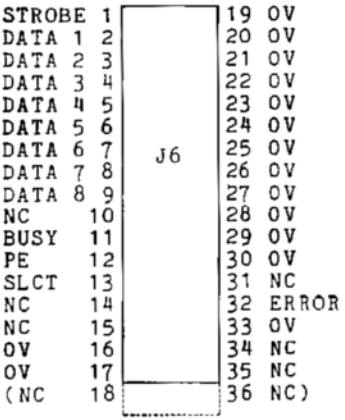
08 ch0	input-VDPINT	out-no connect
09 ch1	input-4MHz/13	out-DART ser clock 0
0A ch2	input-4MHz/13	out-DART ser clock 1
0B ch3	input-CSTTE edge	out-none

0C,0D,0E,0Fh
These are four contiguous read/write ports for the DART.

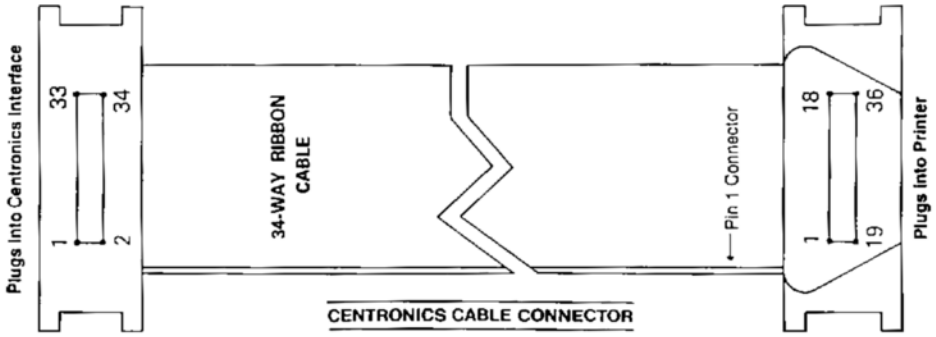
0C	chA data
0D	chB data
0E	chA control
0F	chB control

Ports 10h to 1Eh are currently unused with 1Fh reserved for cassette remote control.
Port addresses 20h upwards will be available as off-board I-O ports in the disc expansion units.

10 PARALLEL PRINTER INTERFACE

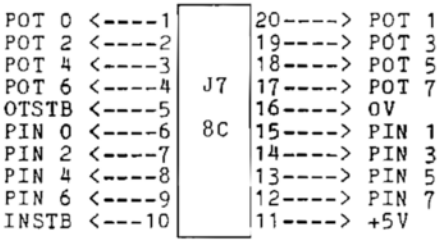


MTX500 Series Centronics Type Parallel Printer I/F Connector
34-Way (17+17) Right Angle Header Plug



11 PARALLEL INPUT/OUTPUT PORT

This is an uncommitted TTL compatible PIO and uses port 7, and is available on an internal 20 pin DIL socket. The port is normally transparent but input data may be latched by taking INSTB to a logic low. The output port is normally tri-state but may be made active by taking OTSB to a logic low. Only TTL compatible signals may be used. The 5V current drain must not exceed 20mA.



12 MEMOTECH DMX80 PARALLEL PRINTER CONNECTOR

SIGNAL PIN No.	RETURN PIN No.	SIGNAL	DIRECTION
1	19	STROBE	IN
2	20	DATA 1	IN
3	21	DATA 2	IN
4	22	DATA 3	IN
5	23	DATA 4	IN
6	24	DATA 5	IN
7	25	DATA 6	IN
8	26	DATA 7	IN
9	27	DATA 8	IN
10	28	ACKNLG	OUT
11	29	BUSY	OUT
12	30	PE	OUT
13	-	SLCT	OUT
14	-	AUTO FEED XT	IN
15	-	NC	-
16	-	CHASSIS-GND	-
18	-	NC	-
19-30	-	GND	-
31	-	INIT	IN
32	-	ERROR	OUT
33	-	GND	-
34	-	NC	-
35	-	-	-
36	-	SLCT-IN	IN

Printed in Great Britain by Butler and Tanner Ltd, Frome and London.

MEMOTECH

Memotech Limited, Witney, Oxon OX8 6BX